Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Вебзастосунок для запису пацієнтів до лікаря з персоналізованим інтерфейсом
Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.230132.01.03.ПЗ

Виконав студент IV курсу, група ІПЗс-23-1 _____ Владислав ВАСІЛЬЄВ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник канд. тех. наук, доцент _____ Оксана ОНИШКО
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. тех. наук, доцент _____ Юрій ФОРКУН
Посада Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

3 червня 2026 р.

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Васільєв Владислав Ігорович

Прізвище, ім'я, по батькові студента

1. Тема роботи Вебзастосунок для запису пацієнтів до лікаря з персоналізованим інтерфейсом

Керівник роботи Онишко Оксана Григорівна, кандидат технічних наук, доцент

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 13.02.2026 р.
2. Строк подання студентом роботи на кафедру 01.06.2026 р.
3. Вихідні дані до роботи Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області, проєктування інформаційної системи, реалізація та тестування.
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
1. Діаграма класів
2. Діаграма варіантів використання
3. Діаграма послідовності

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Форкун Ю. В., канд. техн. наук, доцент	05.05.26	25.05.26
Антиплагіат	Форкун Ю. В., канд. техн. наук, доцент	05.05.26	25.05.26

7. Дата видачі завдання « 02 » січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1. Ознайомлення з тематикою дипломного проектування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	01.12– 31.12.2025	
2. Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2026	
3. Проектування програмного забезпечення	21.02 – 20.03 2026	
4. Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2026	
5. Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2026	
6. Попередній захист КвР	травень 2026	
7. Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.06.2026	
8. Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2026	

Студент _____ В.І. Васільєв
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ О.Г. Онишко
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи: Вебзастосунок для запису пацієнтів до лікаря з персоналізованим інтерфейсом

Автор проєкту: Васільєв Владислав Ігорович

Керівник проєкту: Онишко Оксана Григорівна.

Пояснювальна записка: 65 с., 13 рис., 1 табл., 4 дод., 38 джерел.

Графічна частина: 3 креслення формату А4.

ВЕБЗАСТОСУНОК, LARAVEL, PHP, TYPESCRIPT, МЕДИЧНА ІНФОРМАЦІЙНА СИСТЕМА, АВТОМАТИЗАЦІЯ ЗАПИСУ, ПЕРСОНАЛІЗАЦІЯ ІНТЕРФЕЙСУ, БАЗА ДАНИХ, MVC, REST API.

Мета кваліфікаційної роботи: розробка сучасного програмного рішення для медичних закладів, що дозволяє автоматизувати процес запису пацієнтів на прийом, забезпечує зручне управління графіком лікарів та підвищує якість взаємодії з користувачем через персоналізований інтерфейс. У роботі проведено дослідження актуальних проблем цифровізації медичних послуг, зокрема недоліків існуючих систем онлайн-запису, таких як складність інтерфейсів та низька адаптивність графіків роботи спеціалістів. Для реалізації рішення було обрано екосистему PHP та фреймворк Laravel, що забезпечує високу надійність архітектури та безпеку медичних даних.

Практичне значення роботи полягає у створенні готового до впровадження інструменту, який дозволяє медичним центрам оптимізувати навантаження на персонал та мінімізувати вплив людського фактора при формуванні черги. Персоналізований інтерфейс підвищує доступність сервісу для різних груп пацієнтів, що сприяє покращенню загального досвіду користування медичними послугами в цифровому форматі.

24.05.2026
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.230132.01.03.ПЗ	Пояснювальна записка	65		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
			Графічні документи			
4	А3	КвРІПЗс.230134.01.03.Е8	Діаграма варіантів	1		
			використання			
5	А3	КвРІПЗс.230134.01.03.Е8	Діаграма класів	1		
6	А3	КвРІПЗс.230134.01.03.Е8	Діаграма послідовності	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.230132.01.03.ПЗ		
Виконав		Васільєв В.І.		25.04	Вебзастосунок для запису пацієнтів до лікаря з персоналізованим інтерфейсом	Літ.	Арк. 1 / Аркушів 1
Керівник		Онишко О.Г.		25.04			
Н. контр.		Форкун Ю.В.		25.04		ХНУ, ІПЗс-23-1	
Зав. каф.		Бедратюк Л.П.		25.04			

ЗМІСТ

ВСТУП

Сучасний стан розвитку інформаційних технологій характеризується стрімкою цифровою трансформацією всіх сфер суспільного життя, серед яких охорона здоров'я посідає одне з найбільш пріоритетних місць. Глобальна тенденція до переходу від паперового документообігу до інтегрованих медичних інформаційних систем вимагає розробки якісно нових інструментів взаємодії між лікарем та пацієнтом. У центрі цієї еволюції знаходиться процес автоматизації надання медичних послуг, де першочерговим етапом стає зручне та ефективне планування візитів. Актуальність обраної теми дослідження зумовлена необхідністю подолання бар'єрів між пацієнтом та фахівцем шляхом створення інтуїтивно зрозумілих вебзастосунків, які мінімізують час на адміністративні процедури та забезпечують цілодобову доступність медичної допомоги.

На сьогоднішній день медичні заклади стикаються з серйозними викликами у сфері менеджменту потоків пацієнтів, що безпосередньо впливає на якість обслуговування та рівень довіри громадян до системи охорони здоров'я. Традиційні методи реєстрації через телефонний зв'язок або безпосереднє відвідування реєстратури поступово втрачають свою ефективність через високу ймовірність виникнення помилок, спричинених людським фактором, та обмеженість у часі. Впровадження автоматизованої системи запису дозволяє не лише розвантажити медичний персонал, але й забезпечити пацієнтам прозорий механізм вибору спеціаліста, ознайомлення з переліком послуг та самостійного планування візиту у відповідності до актуального графіка лікаря. Саме тому розробка персоналізованого веб-інтерфейсу, який враховує індивідуальні потреби користувача та специфіку роботи конкретної медичної установи, є стратегічно важливим завданням для розвитку галузі.

Теоретичне підґрунтя роботи базується на аналізі сучасних архітектурних патернів побудови вебзастосунків, зокрема об'єктно-орієнтованого підходу та архітектури MVC. Використання фреймворку Laravel для реалізації серверної

логіки дозволяє забезпечити високий рівень безпеки даних, що є критично важливим при роботі з конфіденційною інформацією про стан здоров'я пацієнтів. Важливим аспектом дослідження є також інтеграція типізованих мов програмування, таких як TypeScript, на стороні клієнта, що дозволяє суттєво підвищити надійність коду та покращити динамічну взаємодію користувача з елементами інтерфейсу. Особлива увага приділяється алгоритмам перевірки доступності часових слотів, які повинні працювати в режимі реального часу, виключаючи будь-які конфлікти в розкладі лікарів.

Об'єктом дослідження є процеси автоматизації надання медичних послуг у межах спеціалізованих вебзастосунків. Предметом дослідження виступають програмні методи та алгоритмічні рішення для реалізації системи онлайн-запису пацієнтів з акцентом на персоналізацію інтерфейсу користувача та оптимізацію бекенд-логіки. В ході написання роботи було проаналізовано існуючі аналоги на ринку медичного програмного забезпечення, виявлено їхні переваги та недоліки, що дозволило сформулювати вимоги до нового, більш гнучкого продукту. Окремим напрямком роботи є дослідження принципів адаптивного дизайну, який гарантує стабільну роботу системи на широкому спектрі пристроїв, від персональних комп'ютерів до мобільних телефонів, що є ключовим показником доступності сучасного веб-сервісу.

Наукова новизна отриманих результатів полягає в удосконаленні архітектурного підходу до побудови систем запису, де клієнтська частина безпосередньо взаємодіє зі складними структурами графіків роботи через оптимізовані API-запити. Запропонована модель фільтрації спеціалістів та послуг дозволяє пацієнту максимально швидко знайти необхідне рішення, спираючись на динамічні дані, що оновлюються без перезавантаження сторінки. Практичне впровадження розробленого рішення в діяльність медичних центрів дозволить підвищити економічну ефективність установ за рахунок кращої завантаженості лікарів та зменшення кількості пропущених візитів завдяки системі сповіщень та зручному візуальному відображенню розкладу.

Структура роботи побудована таким чином, щоб послідовно висвітлити всі етапи життєвого циклу розробки програмного продукту, починаючи від аналізу предметної області та закінчуючи тестуванням готового функціоналу. У розділах пояснювальної записки детально описано процес вибору технологічного стеку, проектування реляційної моделі бази даних, розробку інтерфейсних рішень та впровадження логіки валідації записів. Результати роботи можуть бути використані як база для подальшого розширення системи, зокрема для інтеграції з електронними медичними картками або модулями онлайн-консультацій, що підтверджує масштабованість та актуальність обраного шляху розробки.

Додатковою технологічною перевагою спроєктованої вебсистеми є впровадження наскрізних інженерних методів оптимізації швидкодії на рівні взаємодії між клієнтською частиною та сервером бази даних. Для забезпечення мінімального часу відклику інтерфейсу при генеруванні розгалужених сіток робочого часу лікарів було застосовано механізми агресивного кешування статичних та умовно-динамічних даних, а також оптимізовано структуру реляційних індексів. Це дозволило мінімізувати кількість важких повторних запитів до СУБД та знизити загальне обчислювальне навантаження на апаратні потужності сервера, гарантуючи безперебійне функціонування платформи навіть у періоди пікової активності користувачів під час масового бронювання слотів.

Окрему увагу в межах дослідження приділено питанням інформаційної безпеки та суворого розмежування прав доступу, що є критично важливим критерієм для систем, які оперують конфіденційними персональними даними громадян. Завдяки впровадженню багаторівневої архітектури автентифікації та авторизації, кожен тип користувачів взаємодіє виключно із дозволеним йому набором маршрутів і контролерів, повністю ізолюючи адміністративні функції від публічного контуру системи. Окрім цього, розроблені алгоритми гнучкої валідації вхідних потоків даних на рівні Laravel Form Requests надійно блокують потенційні спроби несанкціонованого втручання в роботу бази даних чи підміни параметрів запису, забезпечуючи високу стійкість вебзастосунку до поширених мережевих загроз та кібератак.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз проблем у сфері

Сучасний етап розвитку глобальної системи охорони здоров'я характеризується стрімким переходом від традиційних паперових та напівавтоматизованих методів управління до повноцінних цифрових екосистем. Проте процес інтеграції інформаційних технологій у повсякденну практику медичних закладів супроводжується низкою системних, технічних та організаційних проблем. Тривалий час взаємодія між пацієнтом та клінікою опиралася на ручну працю реєстраторів або телефонний зв'язок, що в умовах зростання щільності населення та підвищення динаміки суспільного життя виявило свою повну неефективність. Головна деструктивна ознака застарілих підходів полягає у критично високому рівні впливу людського фактора на кожному етапі координації процесів, починаючи від первинного звернення особи і закінчуючи безпосереднім візитом до профільного спеціаліста.

Аналіз поточної ситуації у сфері надання медичних послуг вказує на те, що однією з найгостріших проблем залишається нераціональний розподіл часових ресурсів як медичного персоналу, так і самих пацієнтів. Традиційні системи реєстрації створюють ситуації нерівномірного навантаження на лікарів, коли періоди повної відсутності записів раптово змінюються надмірним потоком відвідувачів. Це призводить до утворення тривалих живих черг, які не лише знижують рівень задоволеності клієнтів, але й створюють додаткове епідеміологічне навантаження на приміщення клінік. Для пацієнта процес запису на прийом через телефонні дзвінки часто перетворюється на тривале очікування на лінії, оскільки пропускна здатність класичних кол-центрів є обмеженою і не здатна впоратися з піковими навантаженнями в ранкові години.

З технічної точки зору, більшість наявних програмних комплексів, які використовуються в медичній сфері, мають закриту або монолітну архітектуру. Це створює суттєві перешкоди для їхньої інтеграції з іншими сучасними сервісами та

мобільними застосунками. Відсутність гнучких інструментів взаємодії з клієнтом призводить до того, що інформація про графіки роботи лікарів оновлюється із суттєвими затримками. Коли лікар змінює свій розклад через непередбачувані обставини, адміністрація закладу часто змушена вручну сповіщати всіх записаних пацієнтів, що вимагає величезних трудовитрат і супроводжується помилками. Через брак автоматизованої синхронізації виникає явище так званих втрачених вікон, коли пацієнт скасовує свій візит, але цей час залишається заблокованим у системі, і інший потенційний клієнт не може ним скористатися.

Важливим аспектом проблеми є архітектурна вразливість систем до явища паралелізму, відомого в інженерії програмного забезпечення як стан гонитви. У періоди високої активності користувачів, коли кілька пацієнтів одночасно намагаються забронювати один і той самий часовий слот через різні інтерфейси, слабкі алгоритми перевірки доступності даних дають збій. Це призводить до виникнення конфліктів подвійного запису, коли на один і той самий час до лікаря призначається кілька людей. Вирішення таких ситуацій у ручному режимі викликає серйозний стрес у медичного персоналу та руйнує репутацію медичного закладу. Відсутність надійного шару ізоляції транзакцій на рівні баз даних є прямим наслідком застарілих архітектурних підходів, які використовуються в медичних інформаційних системах попередніх поколінь.

Окрему групу проблем становить незадовільний стан інтерфейсів користувача в медичних програмних продуктах. Більшість систем створювалися без урахування сучасних стандартів людино-машинної взаємодії, через що вони мають перевантажений, неінтуїтивний вигляд та складну навігацію. Для лікаря, який має обмежений час на приймання одного пацієнта, необхідність здійснювати велику кількість маніпуляцій у програмі для фіксації результатів або перегляду свого розкладу стає додатковим тягарем. Це відволікає спеціаліста від його основних обов'язків та знижує якість надання медичної допомоги. Для пацієнтів старшого віку або осіб з обмеженими можливостями використання таких платформ стає практично неможливим, що обмежує їхній доступ до медичних послуг.

Проблема масштабованості та адаптивності медичного програмного забезпечення також стоїть надзвичайно гостро. Традиційні системи часто розроблялися як локальні десктопні рішення, прив'язані до конкретної фізичної інфраструктури клініки. У сучасних реаліях, коли користувачі вимагають цілодобового доступу до сервісів з будь-яких пристроїв, такі системи демонструють свою повну непридатність. Перенесення таких комплексів у хмарне середовище без докорінної зміни архітектури не вирішує проблему продуктивності. При збільшенні кількості одночасних запитів вебсервери починають демонструвати критичні затримки, а час відклику інтерфейсу зростає до неприпустимих значень, що повністю нівелює переваги автоматизації.

Не менш важливою є проблема безпеки та розмежування прав доступу до персональних даних. Медична інформація належить до категорії конфіденційних даних найвищого рівня та підлягає суворому правовому регулюванню. Проте в багатьох діючих системах механізми контролю доступу реалізовані на поверхневому рівні. Відсутність гнучкої рольової моделі призводить до того, що персонал клініки отримує надлишкові привілеї, які не потрібні для виконання їхніх безпосередніх обов'язків. Це створює серйозні ризики витоку інформації або її навмисного чи випадкового спотворення. Крім того, слабка валідація вхідних даних на рівні користувацьких інтерфейсів відкриває можливості для проведення зловмисниками типових вебвірусних атак, таких як впровадження стороннього коду чи ін'єкції в базу даних.

Глобалізація та розвиток медичного туризму висунули нові вимоги до систем автоматизації, серед яких на перше місце виходить підтримка мультимовності та динамічної локалізації. Велика кількість медичних центрів обслуговує іноземних громадян або працює в регіонах з кількома офіційними мовами. Проте архітектура більшості систем не передбачає гнучкого збереження та виведення текстового контенту різними мовами. Використання статичних файлів локалізації або жорстке кодування текстових констант всередині програмного коду унеможливлює швидке додавання нових мовних версій без залучення розробників і перекомпіляції всієї

системи. Це значно здорожує підтримку продукту та обмежує ринковий потенціал медичного закладу.

Ще одним критичним недоліком є неефективна організація обміну даними між клієнтською та серверною частинами додатків. Традиційні вебсторінки медичних порталів використовують механізм повного перезавантаження контенту при кожній маніпуляції користувача. Це створює надлишкове навантаження на мережеві канали та змушує клієнта очікувати на оновлення всього інтерфейсу навіть при простій зміні дати у календарі. Відсутність асинхронного підходу до зчитування даних та оптимістичного оновлення стану користувацького інтерфейсу створює враження повільної та нестабільної роботи програми, що викликає роздратування у користувачів і змушує їх повертатися до традиційних способів комунікації з клінікою.

Аналіз наявних недоліків у сфері медичного програмного забезпечення чітко вказує на існування серйозного розриву між потребами сучасного медичного бізнесу та можливостями морально застарілих інформаційних систем. Проблема полягає не просто в необхідності перенесення записів з паперових журналів у цифрові бази даних, а у створенні принципово нової архітектури взаємодії. Ця архітектура має забезпечувати абсолютну консистентність даних під високими навантаженнями, гарантувати найвищий рівень безпеки конфіденційної інформації через суворі механізми авторизації, надавати інтуїтивно зрозумілий та швидкодіючий інтерфейс для всіх категорій користувачів, а також підтримувати легку масштабованість і гнучку динамічну локалізацію контенту.

Таким чином, розв'язання окресленого комплексу проблем вимагає розробки інтегрованої вебсистеми, яка базується на передових архітектурних шаблонах та технологіях. Застосування сучасних серверних фреймворків у поєднанні із суворо типізованими мовами програмування на стороні клієнта дозволить створити надійне середовище для координації лікарського прийому. Тільки через повне усунення проміжних ручних ланок, впровадження атомарних транзакцій та ізоляцію бізнес-логіки в окремий сервісний шар можна досягти якісно нового рівня

автоматизації, який мінімізує експлуатаційні витрати медичних установ і забезпечить максимальний комфорт та безпеку для пацієнтів.

1.2 Огляд сучасних технологій для управління записами до лікарів

Управління потоками пацієнтів у сучасному медичному середовищі базується на використанні інтелектуальних систем планування, які виходять далеко за межі простих цифрових журналів. Основною технологічною задачею таких систем є динамічна обробка часових інтервалів (слотів), що враховує не лише графік роботи лікаря, а й специфіку конкретних медичних послуг. Сучасний підхід передбачає використання алгоритмів автоматичного розрахунку тривалості прийому, де кожен тип маніпуляції має свій часовий коефіцієнт. Це дозволяє системі автоматично блокувати необхідну кількість часу в розкладі, запобігаючи виникненню накладок та забезпечуючи рівномірне навантаження на кабінети та обладнання.

Важливим компонентом сучасних систем є технологія асинхронної обробки даних, яка забезпечує миттєву фільтрацію та пошук інформації без переривання взаємодії користувача з інтерфейсом. Впровадження механізмів каскадного вибору (за ланцюжком «Спеціалізація — Лікар — Послуга») дозволяє значно скоротити шлях пацієнта до фінальної мети — підтвердження запису. Такі архітектурні рішення базуються на принципах реактивності, де зміна одного параметра (наприклад, вибір напрямку лікування) миттєво оновлює доступні опції в інших категоріях. Це мінімізує когнітивне навантаження на користувача та знижує відсоток помилок при самостійному бронюванні візитів.

Окремо слід виділити технології персоналізації користувацького досвіду, які включають адаптивне відображення розкладу залежно від пристрою та налаштувань користувача. Використання гнучких сіток (Grids) та компонентних модальних вікон дозволяє відображати складні структури, такі як тижневі графіки роботи лікарів, у компактному та читабельному вигляді навіть на екранах смартфонів. Персоналізація також проявляється у впровадженні функцій

збереження стану (наприклад, вибір темної чи світлої теми, мовні налаштування), що створює комфортне візуальне середовище та сприяє тривалій лояльності пацієнта до сервісу.

На рівні обробки бізнес-логіки ключову роль відіграють методи суворої валідації введених даних. Технології перевірки доступності в реальному часі дозволяють системі аналізувати не лише точний час початку візиту, а й перетин з уже існуючими записами з урахуванням тривалості процедур. Такий підхід гарантує цілісність розкладу та дозволяє реалізувати інтелектуальні підказки: у разі зайнятості обраного слоту система автоматично аналізує вільні «вікна» та пропонує пацієнту найближчий доступний час. У поєднанні з надійними інструментами розробки (такими як Laravel для бекенду та TypeScript для фронтенду), ці підходи дозволяють створити стійку до помилок систему, що є критично важливим для установ з високою інтенсивністю відвідувань.

Нинішній ринок медичних інформаційних систем (MIC) відзначається значною різноманітністю: від масштабних хмарних екосистем до локальних продуктів вузького спрямування. Детальний розгляд популярних аналогів дозволяє стверджувати, що більшість розробок фокусується на комплексному адмініструванні великих клінік. Це неминуче призводить до перевантаження інтерфейсів надлишковим функціоналом, що ускладнює взаємодію для звичайного користувача. З технічної точки зору, чимало систем базуються на застарілих архітектурних моделях. Це перешкоджає швидкій адаптації фронтенд-частини під індивідуальні запити пацієнта чи специфіку роботи лікаря, створюючи комунікаційні бар’єри та знижуючи загальну ефективність цифрової реєстрації у невеликих приватних центрах.

З погляду технологічної реалізації, існуючі сервіси поділяються на кілька ключових груп. До першої належать SaaS-платформи, які забезпечують швидке розгортання, проте суттєво обмежують гнучкість у налаштуванні бізнес-логіки та глибокій персоналізації обробки даних. Друга група охоплює кастомні розробки на базі різноманітних вебфреймворків. Практика показує, що монолітні архітектурні рішення в цій категорії часто демонструють зниження продуктивності при

інтенсивному наповненні бази даних. Натомість продукти, спроектовані на базі прогресивних PHP-інструментів, зокрема Laravel, виявляються значно стабільнішими завдяки ефективним механізмам кешування, чергам задач та модульній структурі, що полегшує налаштування параметрів безпеки та пришвидшує відгук системи.

Важливе значення має оцінка технічних та системних вимог, що висуваються до медичного софту. Більшість актуальних рішень передбачають розміщення у сертифікованій хмарній інфраструктурі з високим показником відмовостійкості (SLA). Дослідження показало, що фундаментальним фактором безпеки є впровадження протоколів HTTPS та шифрування даних на рівні сховищ для захисту приватної інформації. Проте в багатьох продуктах спостерігається дефіцит уваги до адаптивної верстки: мобільні інтерфейси часто є спрощеними репліками десктопних версій, що ігнорують специфіку мобільного досвіду пацієнта.

Підбиваючи підсумок аналізу програмно-технічної бази, можна зробити висновок, що сегмент вузькоспеціалізованих застосунків із акцентом на персоналізацію інтерфейсу досі залишається незаповненим. Популярні системи не завжди забезпечують необхідну прозорість та легкість управління для різних категорій користувачів. У зв’язку з цим, проектування власної системи на базі Laravel із пріоритетом на UX-дизайн та оптимізовану серверу логіку є раціональним кроком. Такий підхід дозволяє поєднати стабільність рішень корпоративного рівня з гнучкістю персоналізованого сервісу, що відповідає актуальним стандартам інженерії програмного забезпечення.

На прикладі ринку медичних послуг міста Хмельницького видно, що первинний етап діджиталізації вже завершено, проте існує значний запит на більш гнучкі інструменти. Більшість закладів міста використовують великі національні платформи, які, хоч і виконують роль посередників, часто залишаються занадто уніфікованими та позбавленими персоналізованого підходу до кінцевого пацієнта. Основними прикладами вебзастосунків, що активно функціонують у Хмельницькому, є:

– Medics (Медікс). Ця система є найбільш розповсюдженою серед комунальних закладів міста, включаючи міські поліклініки та центри первинної медико-санітарної допомоги (Рисунок 1.1). Застосунок дозволяє пацієнтам записуватися на прийом, переглядати електронні направлення та результати аналізів. Однак, користувачі часто відмічають перевантаженість інтерфейсу та складність навігації для людей старшого віку, що підкреслює актуальність розробки більш лаконічних, персоналізованих рішень.

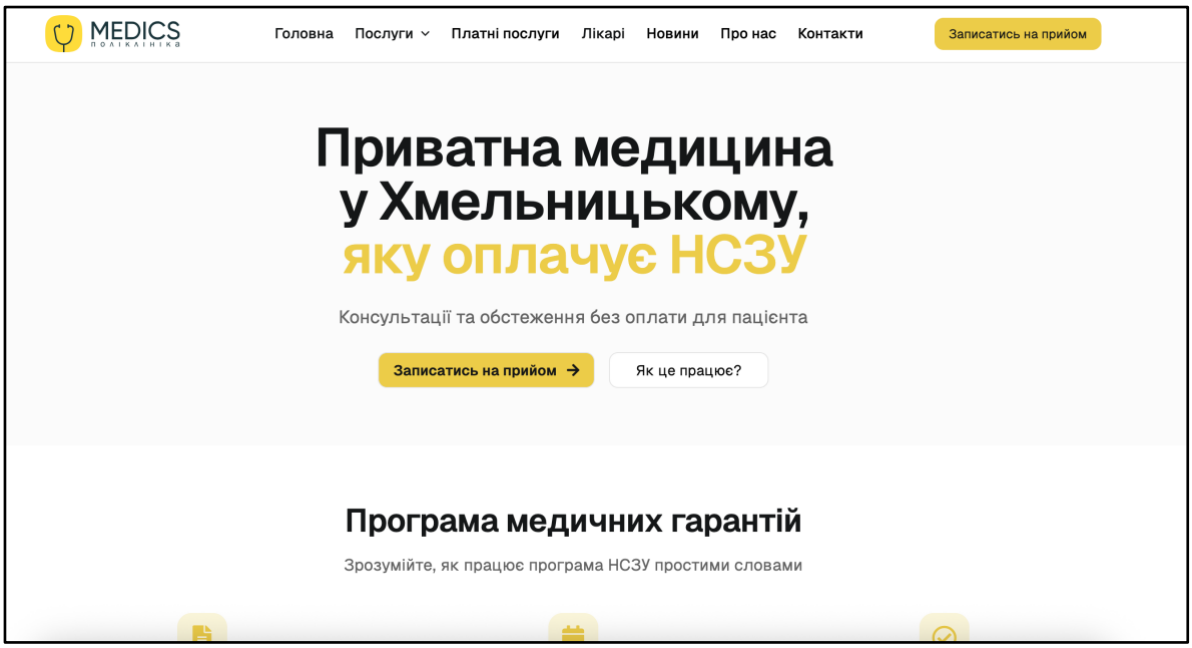


Рисунок 1.1 – головна сторінка сервісу Medics

– Health24 (Хелс 24): Платформа, на яку активно переходять великі медичні установи, такі як Хмельницька обласна лікарня та Хмельницька міська дитяча лікарня (Рисунок 1.2). Вебзастосунок має потужний функціонал для синхронізації з державною системою eHealth, підтримку черг та електронних медкарток. Попри високу технологічність, система орієнтована на стандартизацію, що обмежує можливості кастомізації інтерфейсу під конкретні вузькоспеціалізовані медичні центри, тобто дизайн даного сайту не є актуальним на сьогоднішній час за багатьма пунктами і було б доцільно його оновити, щоб покращити процес користування цим ресурсом.

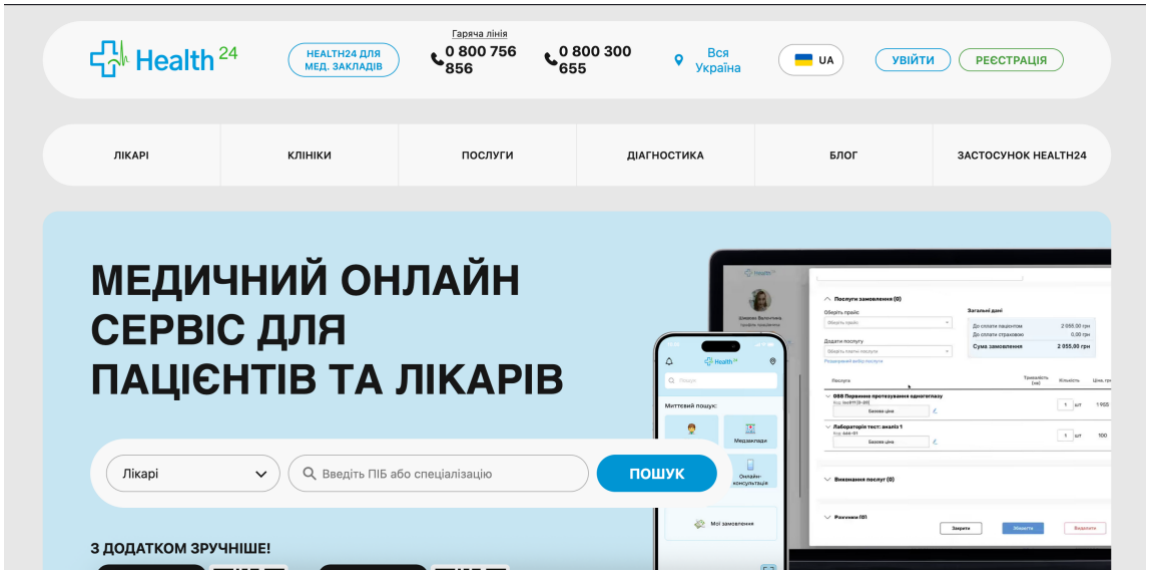


Рисунок 1.2 – головна сторінка сервісу Health24

Аналіз цих систем підтверджує, що попри наявність технічної бази, існує значний попит на "нішеві" застосунки. Наявне ПЗ часто страждає від "універсальності", де інтерфейс пацієнта виглядає однаково для запису до стоматолога і до кардіолога, не враховуючи специфіку збору анамнезу чи попередньої підготовки до візиту. Це створює підґрунтя для мого дипломного проєкту — розробки вебзастосунку, який запропонує індивідуальні сценарії взаємодії (персоналізовані форми запису, кастомні нагадування та адаптивні кабінети), що дозволить невеликим клінікам м. Хмельницький виділитися на фоні стандартизованих сервісів-гігантів.

1.3 Формулювання цілей, завдань і вимог до системи

Успішна реалізація будь-якого складного програмного комплексу в медичній сфері безпосередньо залежить від чіткості визначення його концептуального базису, який закладається на етапі передпроєктного аналізу. Формулювання цільових орієнтирів дозволяє не лише окреслити фінальний вигляд програмного продукту, але й сформувати об'єктивні критерії оцінки ефективності його майбутнього функціонування. Процес розробки інформаційної системи для координації лікарського прийому вимагає інтеграції різнорідних вимог, що

висуваються з боку пацієнтів, медичного персоналу та адміністративного корпусу клініки, що робить задачу проектування багатоаспектною та комплексною.

Головна стратегічна мета виконання даної кваліфікаційної роботи полягає у проєктуванні, програмній реалізації та верифікації високопродуктивної, надійної та масштабованої вебсистеми, призначеної для повної автоматизації процесів пошуку медичних фахівців, перегляду їхньої актуальної доступності та безпосереднього бронювання часових слотів. Досягнення цієї мети дозволить мінімізувати часові витрати користувачів на взаємодію з медичною установою, повністю усунути проміжні ручні ланки у вигляді реєстраторів або операторів кол-центрів, а також підвищити коефіцієнт корисного використання робочого часу лікарів за рахунок оптимізації та рівномірного розподілу їхнього щоденного навантаження. Важливо, що для цього потрібно провести певний аналіз, результатом якого можу бути діаграма використання самого модулю запису до лікаря (Рисунок 1.3).

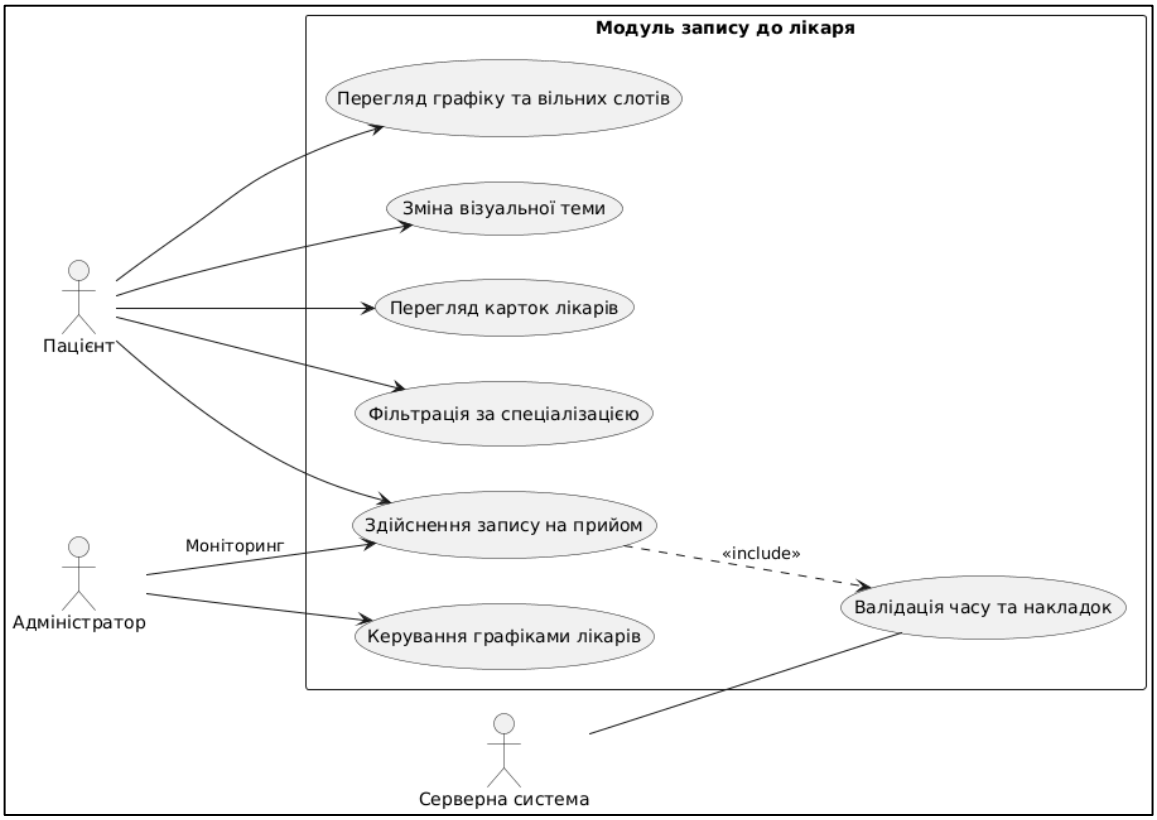


Рисунок 1.3 – Діаграма використання модулю запису до лікаря

Для успішного досягнення детермінованої мети необхідно послідовно вирішити комплекс взаємопов'язаних науково-технічних та інженерних завдань. Першочерговим завданням є здійснення детального аналізу предметної області, виявлення специфіки бізнес-процесів медичного закладу та дослідження наявних на ринку програмних аналогів з метою врахування їхніх архітектурних переваг та технологічних недоліків. Наступним етапом є формалізація повного спектра вимог до системи та розробка цілісної архітектурної моделі, яка включає проектування реляційної структури бази даних, визначення інтерфейсів взаємодії компонентів та вибір оптимальних шаблонів проєктування.

Практичний блок завдань охоплює безпосередню програмну розробку серверного модуля системи на базі сучасного фреймворку, реалізацію бізнес-логіки обчислення вільних вікон розкладу, інтеграцію гнучкої моделі розмежування прав доступу та впровадження механізмів динамічної локалізації контенту. Паралельно з цим постає завдання створення клієнтського інтерфейсу із застосуванням суворо типізованої мови програмування, що гарантує високу інтерактивність, швидкість відклику та коректність відображення даних на будь-яких типах пристроїв. Заключний етап завдань передбачає проведення всебічного автоматизованого та функціонального тестування створеного програмного забезпечення, виявлення та усунення дефектів, а також підбиття підсумків щодо його практичної цінності та перспектив подальшого масштабування.

Важливим кроком проєктування є деталізація функціональних вимог, які описують безпосередні можливості, що надаються програмним забезпеченням для різних категорій користувачів. Система повинна забезпечувати безперешкодну реєстрацію та автентифікацію пацієнтів, надавати розширені інструменти фільтрації та пошуку лікарів за спеціалізаціями чи медичними послугами, а також візуалізувати актуальний графік роботи у вигляді динамічної сітки часових слотів. З боку адміністративної панелі обов'язковим є наявність інструментарію для гнучкого налаштування базових розкладів, внесення винятків у графіки, управління каталогом послуг та редагування мовних констант сайту без втручання у вихідний код проєкту.

Нефункціональні вимоги до системи визначають якісні характеристики програмного продукту, серед яких на перше місце виходять продуктивність, надійність та безпека. Час генерації та віддачі сторінки або відповіді на API-запити при зчитуванні графіків не повинен перевищувати критичного порогу у триста мілісекунд, що забезпечується впровадженням агресивного кешування на рівні сервера. Надійність системи визначається її здатністю підтримувати абсолютну консистентність даних під час пікових навантажень, що вимагає використання суворих рівнів ізоляції транзакцій СУБД для повного унеможливлення виникнення конфліктів подвійного бронювання одного часового слота різними клієнтами.

Вимоги до безпеки та захисту інформації мають фундаментальне значення, оскільки система оперує персональними та медичними даними, що підлягають суворому законодавчому регулюванню. Програмний комплекс повинен базуватися на надійній моделі контролю доступу на основі ролей, яка гарантує, що жоден користувач не зможе отримати доступ до адміністративних маршрутів або чужих записів без відповідних підтверджених привілеїв. Крім того, архітектура системи має передбачати обов'язкову всебічну валідацію та санітарію всіх вхідних даних на серверному рівні, що блокує потенційні спроби проведення зловмисних атак, таких як впровадження сторонніх SQL-скриптів або міжсайтовий скриптинг.

З технічної та експлуатаційної точок зору, система повинна мати високий рівень адаптивності та кросбраузерності, стабільно функціонуючи у всіх сучасних веб-оглядачах та на мобільних платформах без втрати юзабіліті. Архітектура коду на стороні клієнта має бути спроєктована таким чином, щоб мінімізувати кількість мережевих запитів та об'єм завантажуваних даних, що досягається за рахунок асинхронного обміну інформацією та модульної збірки компонентів. Дотримання розробленого комплексу цілей, завдань та вимог є обов'язковою умовою для створення конкурентоспроможного програмного забезпечення, здатного ефективно вирішити проблеми автоматизації медичної сфери.

Системні вимоги до вебзастосунку фокусуються на забезпеченні високої швидкості відгуку серверної частини та стабільній роботі клієнтського інтерфейсу на пристроях із різними типами екранів. Використання фреймворку Laravel на

стороні сервера має гарантувати високу пропускну здатність та стійкість системи до пікових навантажень, тоді як інтеграція TypeScript на стороні клієнта повинна забезпечити надійність логіки та запобігання помилкам під час виконання скриптів. Поєднання цих вимог у межах єдиного продукту дозволить створити інструмент, який відповідає високим стандартам сучасної інженерії програмного забезпечення та пропонує реальне вирішення актуальних проблем автоматизації медичної логістики.

Окрему увагу на етапі специфікації вимог приділено забезпеченню високої масштабованості системи та її адаптивності до можливих змін у бізнес-процесах медичних установ. Програмний комплекс повинен мати модульну структуру, яка дозволяє безперешкодно розширювати функціонал — наприклад, додавати модулі онлайн-оплати послуг, телемедичних консультацій чи інтеграції з державними реєстрами — без необхідності кардинальної перебудови наявного ядра системи. Важливою технологічною вимогою є контейнеризація всього середовища розробки та розгортання, що гарантує ідентичність роботи програми на локальних машинах розробників, тестових серверах та в продукційній хмарі, суттєво знижуючи витрати на технічну підтримку та адміністрування платформи.

Крім того, система має відповідати суворим ергономічним вимогам та стандартам доступності інтерфейсів, оскільки кінцевими користувачами вебпродукту є люди з різним рівнем комп'ютерної грамотності та фізичних можливостей. Користувацький інтерфейс повинен будуватися на принципах інтуїтивної навігації, де для здійснення ключової дії — запису до лікаря — пацієнту потрібно зробити мінімальну кількість кліків. Впровадження механізмів динамічного перемикання колірних тем та оптимізація контрастності елементів інтерфейсу дозволяють забезпечити комфортну роботу із системою у будь-який час доби та зробити її доступною для осіб із порушеннями зору, що повністю задовольняє сучасні соціальні та технічні стандарти проектування цивільних інформаційних систем.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Архітектурні рішення: обґрунтування модульної структури та компонентного підходу

Проєктування архітектури вебзастосунку для медичних установ є критично важливим етапом, що визначає здатність системи до масштабування, підтримки та безпечної обробки даних. Для реалізації модуля запису пацієнтів було обрано багаторівневу архітектуру, яка базується на поєднанні класичного патерна MVC (Model-View-Controller) на стороні сервера та компонентно-орієнтованого підходу на стороні клієнта. Такий вибір дозволяє чітко розмежувати зону відповідальності між бізнес-логікою, сховищем даних та інтерфейсом користувача, що суттєво полегшує процес розробки та подальшого тестування окремих модулів системи.

Використання фреймворку Laravel зумовлює застосування сервісно-орієнтованого підходу в межах серверної частини. Модульна структура бекенду дозволяє винести складні алгоритми, такі як валідація часових слотів та перевірка конфліктів у розкладі, в окремі класи-сервіси. Це забезпечує повторне використання коду та чистоту контролерів, які виконують лише роль посередників між запитами користувача та логікою системи. На рівні бази даних архітектура підтримує реляційні зв’язки, що гарантує цілісність інформації про лікарів, пацієнтів та їхні візити, а використання Eloquent ORM дозволяє абстрагуватися від складних SQL-запитів, підвищуючи безпеку через автоматичний захист від ін’єкцій.

Клієнтська частина системи спроєктована за принципом модульності інтерфейсних елементів. Кожен блок (картка лікаря, модальне вікно графіка, форма вибору послуг) розглядається як незалежна одиниця, що має власну логіку поведінки, реалізовану на TypeScript. Гібридний характер архітектури виявляється у поєднанні класичного рендерингу сторінок із динамічними асинхронними оновленнями через API. Це дозволяє користувачеві взаємодіяти з розкладом та фільтрами без повного перезавантаження сторінки, що критично важливо для збереження контексту та зручності використання на мобільних пристроях.

Важливим аспектом обраної архітектури є персоналізація на рівні фронтенду, яка реалізується через незалежний модуль управління станом інтерфейсу. Завдяки використанню змінних середовища та препроцесора SCSS, система дозволяє миттєво змінювати візуальні теми та адаптувати сітку під різні роздільні здатності екрана без втручання в логіку бекенду. Такий модульний підхід забезпечує високу відмовостійкість: вихід з ладу або технічне обслуговування одного модуля (наприклад, системи сповіщень) не впливає на працездатність основної функції — перегляду доступних лікарів та здійснення запису. З ціллю полегшення аналізу та подальшої розробки, було побудовано діаграму із загальною архітектурною системою (Рисунок 2.1).

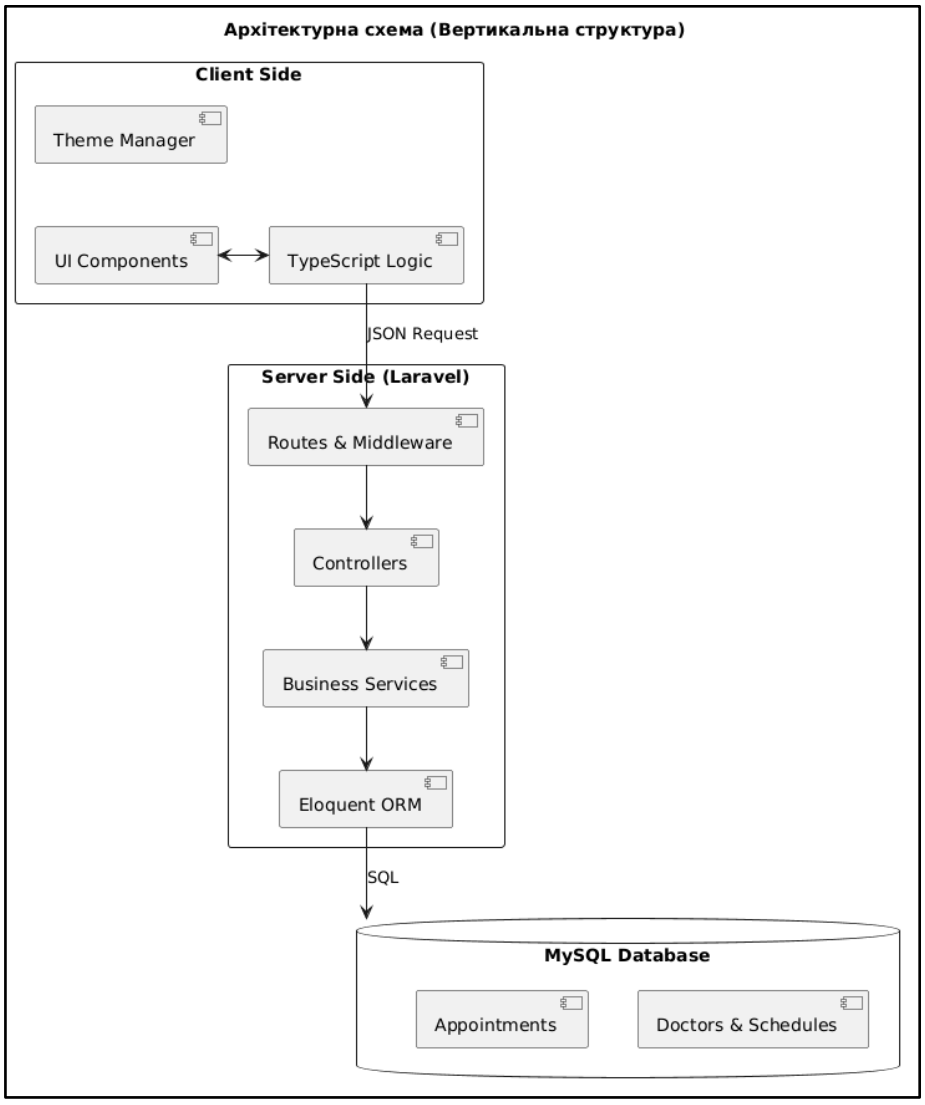


Рисунок 2.1 – загальна архітектурна схема

Важливою особливістю обраного архітектурного підходу є впровадження принципу "Single Source of Truth" (єдиного джерела істини) для управління станом медичних даних. Це реалізується через централізовану обробку запитів на серверному рівні, де кожен вхідний запит проходить крізь систему посередників (Middleware). Такий механізм дозволяє уніфікувати перевірку прав доступу та забезпечити цілісність сесій користувачів перед тим, як дані потраплять до основної бізнес-логіки. Модульність системи також проявляється у використанні сервіс-контейнерів Laravel, що дозволяє реалізувати механізм впровадження залежностей (Dependency Injection). Це значно підвищує гнучкість архітектури: наприклад, за потреби змінити логіку відправки сповіщень пацієнтам з електронної пошти на SMS-повідомлення, розробнику достатньо замінити відповідну реалізацію сервісу, не змінюючи при цьому код контролерів.

На рівні взаємодії клієнта та сервера архітектура використовує оптимізований протокол обміну даними у форматі JSON. Замість передачі великих масивів HTML-розмітки, сервер повертає лише необхідні набори даних про структуру графіку, які потім обробляються на стороні клієнта за допомогою TypeScript-логіки. Такий розподіл обчислювального навантаження дозволяє суттєво пришвидшити роботу інтерфейсу, особливо в умовах нестабільного мобільного з’єднання. Гібридна структура також включає механізм "сплячої" валідації на фронтенді, яка попередньо перевіряє коректність заповнення форм ще до моменту відправки даних на сервер. Це зменшує кількість зайвих мережевих запитів та забезпечує миттєвий зворотний зв’язок із пацієнтом, що є ключовим показником якісного UX-дизайну.

Окрему увагу в архітектурному проектуванні приділено системі персоналізації візуального середовища, яка інтегрована безпосередньо в структуру стилів. Використання SCSS-міксинів та CSS-змінних дозволяє системі динамічно адаптувати колірну палітру та типографіку без необхідності повторної компіляції активів на сервері. Це досягається шляхом збереження преференцій користувача у локальному сховищі браузера (LocalStorage) та їх миттєвого зчитування під час ініціалізації DOM-дерева. Така архітектурна деталь забезпечує безшовне

перемикання між темами, запобігаючи мерехтінню екрана та зберігаючи комфортний рівень зорового сприйняття для пацієнта в різний час доби.

Завершальним елементом модульної структури є система обробки виняткових ситуацій та логування подій. Архітектура передбачає перехоплення помилок на кожному рівні — від невдалих спроб запису в базу даних до помилок парсингу JSON у фронтенд-частині. Це дозволяє системі залишатися стабільною навіть при некоректних діях користувача: замість критичної зупинки роботи застосунок видає інформативне повідомлення та пропонує можливі варіанти вирішення проблеми. Таким чином, обраний гібридний підхід та модульна організація системи створюють надійний фундамент для побудови сучасного медичного сервісу, який здатний ефективно функціонувати в умовах постійно зростаючої кількості користувачів та обсягів медичної інформації.

2.2 Опис структури даних та моделі бази даних

Ефективність функціонування медичного вебзастосунку безпосередньо залежить від якості проєктування схеми бази даних, яка повинна забезпечувати цілісність інформації та швидкість виконання запитів у режимі реального часу. Для реалізації модуля запису було обрано реляційну модель керування даними на базі СУБД MySQL. Вибір реляційного підходу обумовлений необхідністю встановлення суворих зв’язків між сутностями, що дозволяє уникнути дублювання даних та забезпечує можливість побудови складних вибірок, таких як фільтрація лікарів за спеціалізаціями або пошук вільних часових слотів для конкретної дати.

Центральною сутністю системи є модель Doctors, яка містить персональну та професійну інформацію про медичний персонал. Окрім базових полів (ідентифікатор, прізвище, ім’я), таблиця включає посилання на спеціалізацію та текстові описи досвіду роботи. Для забезпечення високої швидкості пошуку, поля, що відповідають за спеціалізацію, є індексованими. Це дозволяє системі миттєво повертати результати при використанні каскадних фільтрів на клієнтській стороні,

що суттєво покращує користувацький досвід та знижує навантаження на сервер під час пікових звернень.

Аби спростити розробку було побудовано базову діаграму класів, яка охоплює лише ключові модулі для цього проєкту (Рисунок 2.2).

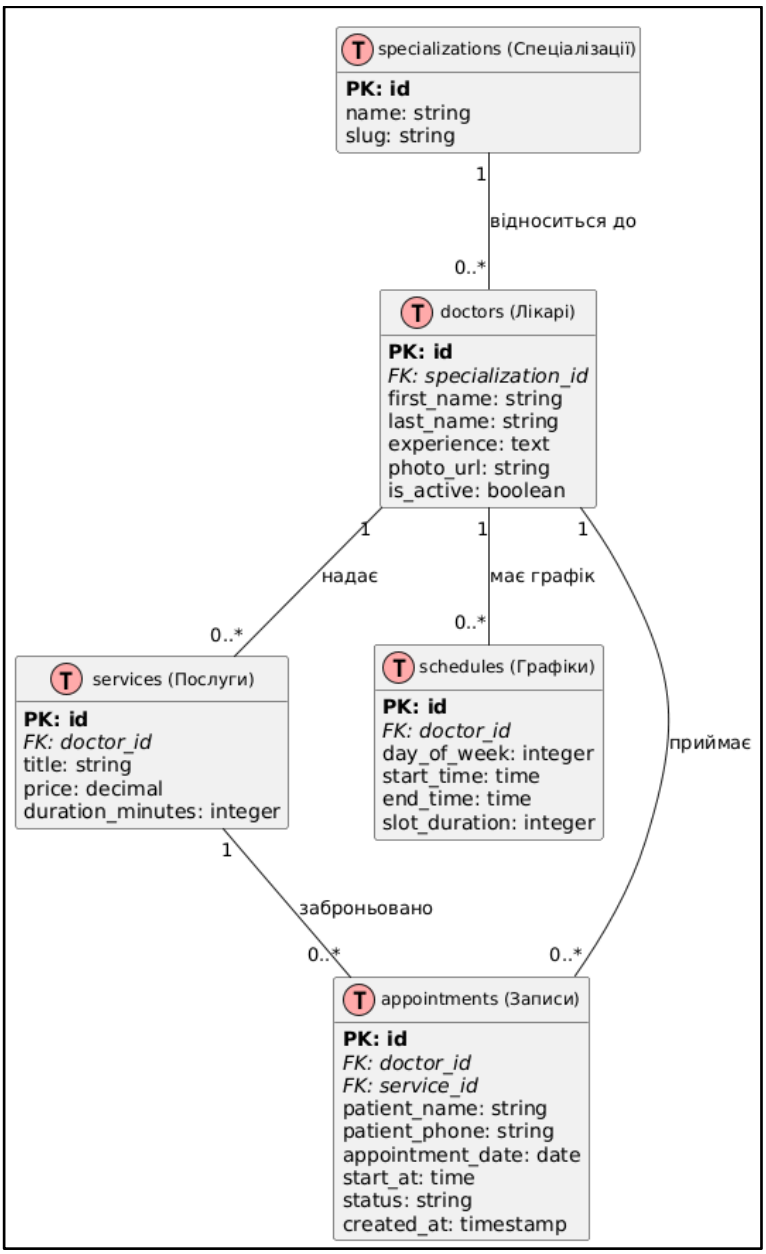


Рисунок 2.2 – діаграма класів для основних модулів

Логіка управління часом реалізована через сутність Schedules, яка перебуває у зв’язку «один до багатьох» із таблицею лікарів. Дана модель описує шаблони робочого часу, включаючи час початку та завершення зміни, а також тривалість стандартного прийому. Такий підхід дозволяє гнучко налаштовувати графік для

кожного фахівця окремо, враховуючи перерви та вихідні дні. Програмна реалізація цієї моделі в екосистемі Laravel через Eloquent ORM дозволяє автоматично підвантажувати пов’язані графіки під час перегляду профілю лікаря, що мінімізує кількість запитів до бази даних за допомогою механізму "eager loading".

Ключовою сутністю для бізнес-логіки є таблиця Appointments, яка фіксує факт бронювання візиту. Вона пов’язує пацієнта, лікаря та конкретну послугу, зберігаючи точну дату та час початку прийому. Важливим аспектом проєктування цієї сутності є використання спеціалізованих типів даних для зберігання часових міток, що дозволяє виконувати точні математичні порівняння на рівні бази даних. Саме ця таблиця використовується сервісом валідації для перевірки накладок: перед збереженням нового запису система виконує запит на пошук існуючих бронювань у заданому часовому діапазоні для обраного лікаря, забезпечуючи атомарність операції запису.

Додатково структура даних включає допоміжні сутності, такі як Specializations та Services, які дозволяють нормалізувати базу даних та уникнути надлишковості. Такий розподіл інформації дозволяє легко масштабувати систему: наприклад, при додаванні нових напрямків лікування або зміні переліку послуг не потрібно змінювати структуру основних таблиць. Загалом, спроєктована модель бази даних є оптимальною для медичного застосунку середнього масштабу, оскільки вона поєднує в собі строгу логічну структуру, високу швидкість обробки транзакцій та гнучкість для майбутніх оновлень функціоналу.

Окремим аспектом проєктування структури даних є реалізація механізму збереження цілісності при видаленні об'єктів. Для забезпечення стабільності системи та запобігання появі «сирітських» записів (записів, що посилаються на неіснуючі об’єкти), у базі даних впроваджено каскадні обмеження на рівні зовнішніх ключів (Foreign Keys). Це означає, що при видаленні профілю лікаря система автоматично обробляє пов'язані з ним графіки та записи, або, що є більш доцільним для медичної сфери, використовує механізм «м’якого видалення» (Soft Deletes). Такий підхід дозволяє приховати дані з інтерфейсу користувача,

зберігаючи їх фізично в базі для архівних цілей або фінансової звітності клініки, що відповідає вимогам до зберігання медичної документації.

Важливим елементом моделі даних є також логування статусів записів. Таблиця Appointments містить поле стану, яке дозволяє відстежувати життєвий цикл візиту: від очікування підтвердження до завершення або скасування прийому. Це дає можливість будувати аналітичні звіти про завантаженість клініки та ефективність роботи кожного окремого лікаря. Загалом, розроблена структура даних не лише відповідає поточним функціональним вимогам вебзастосунку, а й закладає міцний фундамент для майбутньої інтеграції з автоматизованими системами обліку ліків або модулями електронних рецептів, забезпечуючи високу масштабованість та стабільність програмного продукту на всіх етапах його експлуатації.

2.3 Розробка інтерфейсу користувача та UX-рішень

Проєктування інтерфейсу користувача (UI) та розробка логіки взаємодії (UX) для медичного вебзастосунку вимагали особливої уваги до деталей, що впливають на швидкість прийняття рішень пацієнтом. Основним завданням при розробці візуальної концепції стало створення чистого, мінімалістичного середовища, яке не перевантажує користувача зайвими елементами, але водночас надає вичерпну інформацію для здійснення запису. В основу дизайну покладено принцип «mobile-first», що гарантує ідеальну роботу сервісу на смартфонах, оскільки значна частина записів здійснюється саме через мобільні пристрої.

Головна сторінка вебзастосунку виконує роль вхідної точки, де пацієнт отримує перше враження про клініку та її спеціалістів (Рисунок 2.3). Вона спроєктована таким чином, щоб акцентувати увагу на заклику до дії (Call to Action) — пошуку лікаря та перегляду послуг. Використання м’якої колірної палітри з переважанням синіх та білих відтінків підсвідомо формує відчуття стерильності, надійності та професіоналізму, що є критичним для медичної сфери.

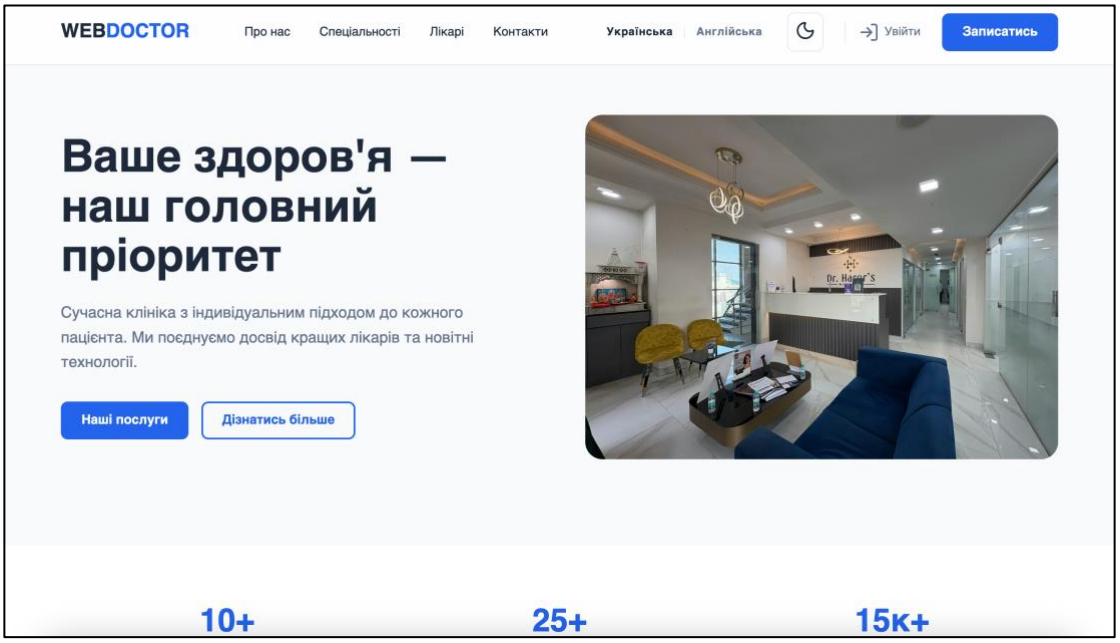


Рисунок 2.3 – вигляд головної сторінки

Розділ «Про нас» та сторінка переліку фахівців реалізовані через сітку карток (Рисунок 2.4), кожна з яких містить лаконічну інформацію про досвід та спеціалізацію лікаря. Для покращення UX було впроваджено каскадні фільтри, які дозволяють пацієнту за два кліки відсіяти непотрібні категорії.

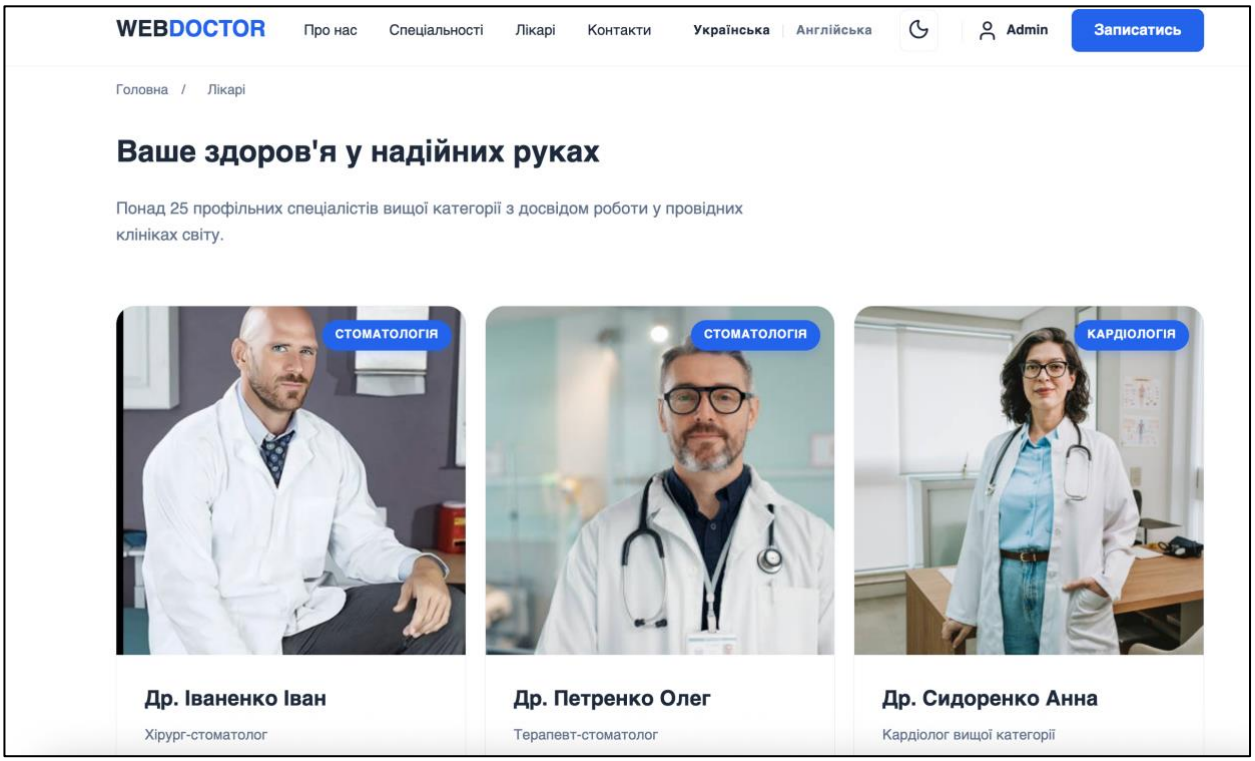


Рисунок 2.4 – вигляд сторінки лікарів

Важливим елементом персоналізації є модуль перемикання тем, інтегрований у шапку сайту. Завдяки використанню змінних SCSS, перехід між світлим та темним режимами відбувається миттєво, забезпечуючи комфортний перегляд у будь-який час доби (Рисунок 2.5).

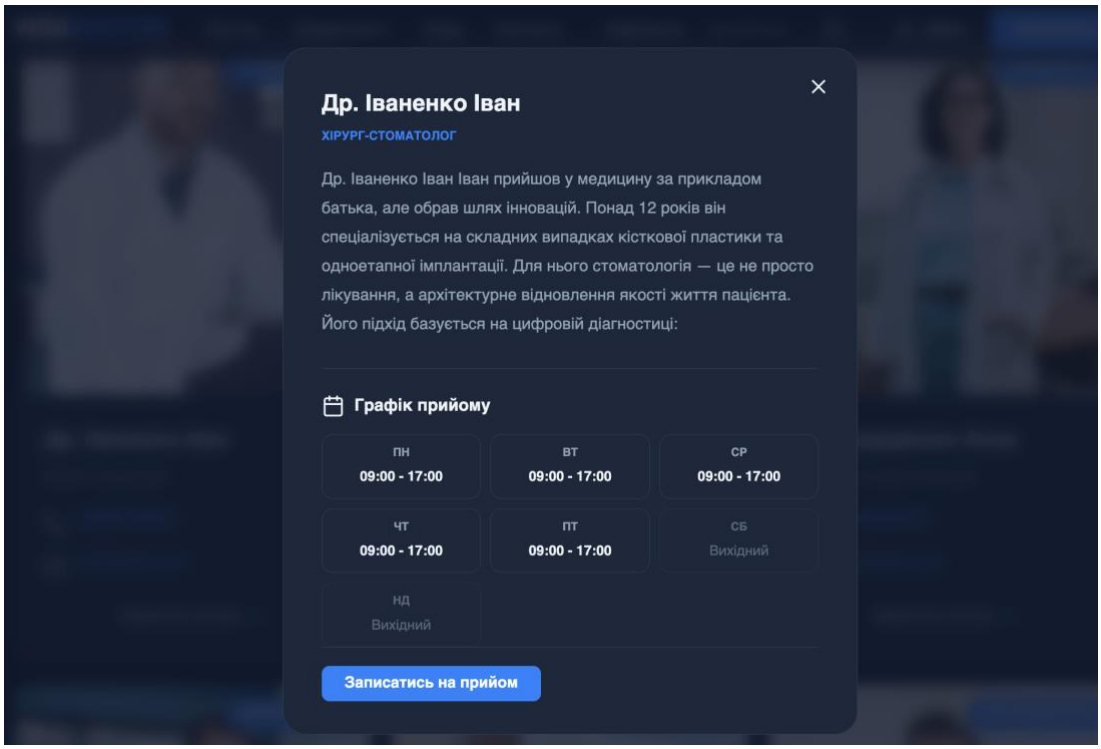


Рисунок 2.5 – вигляд вікна з інформацією про лікаря з темною темою

Основним функціональним вузлом є модуль запису на прийом до лікаря. На цій сторінці є можливість обрати бажаного лікаря, попередньо відфільтрувавши по спеціальності та є можливість вибору часових слотів, реалізований у вигляді інтерактивного календаря або списку доступних годин у модальному вікні. Тут застосовано реактивний підхід: при зміні дати або типу послуги список доступних вікон оновлюється без перезавантаження сторінки через AJAX-запити. Це створює відчуття цілісності застосунку та значно скорочує час очікування. Валідація даних відбувається безпосередньо в момент вибору, що мінімізує ризик отримання відмови від сервера після заповнення всієї форми.

Сторінка особистого кабінету є доступною лише для авторизованих користувачів, в іншому випадку сайт перенаправить на сторінку авторизації або реєстрації (Рисунок 2.6).

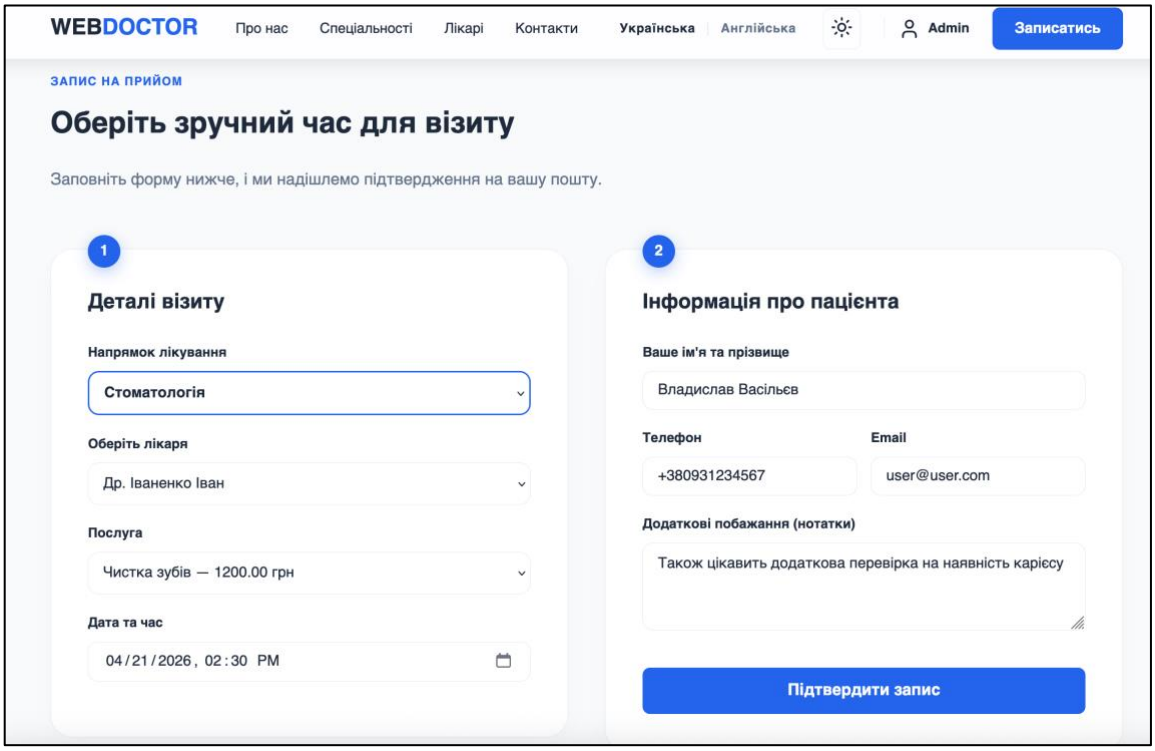


Рисунок 2.6 – вигляд сторінки запису на прийом

Основна мета цього модулю є збір загальної інформації про потенційного пацієнту, а саме ім’я, прізвище, номер телефону, дата народження, адреса та електронна пошта. Ця інформацію може стати в нагоді з організаційних або медичних цілей. Додатково сторінку особистого кабінету користувача можна використовувати як панель моніторингу, де пацієнт може бачити історію своїх візитів, поточні записи та персональні налаштування. Архітектура кабінету максимально спрощена для швидкого доступу до інформації про майбутній прийом. Впровадження індикаторів статусу (наприклад, «Підтверджено», «Очікується», «Завершено») дозволяє користувачеві легко орієнтуватися у стані своїх запитів. Цей функціонал дозволяє користувачу легко орієнтуватись в своїх поточних записах і зменшує ймовірність забутись або пропустити візит до обраного лікаря, а також завжди знати актуальну інформацію з приводу своїх записів до лікарів (Рисунок 2.7).

Також в цьому модулі можна побачити інформацію про попередні записи, які вже пройшли.

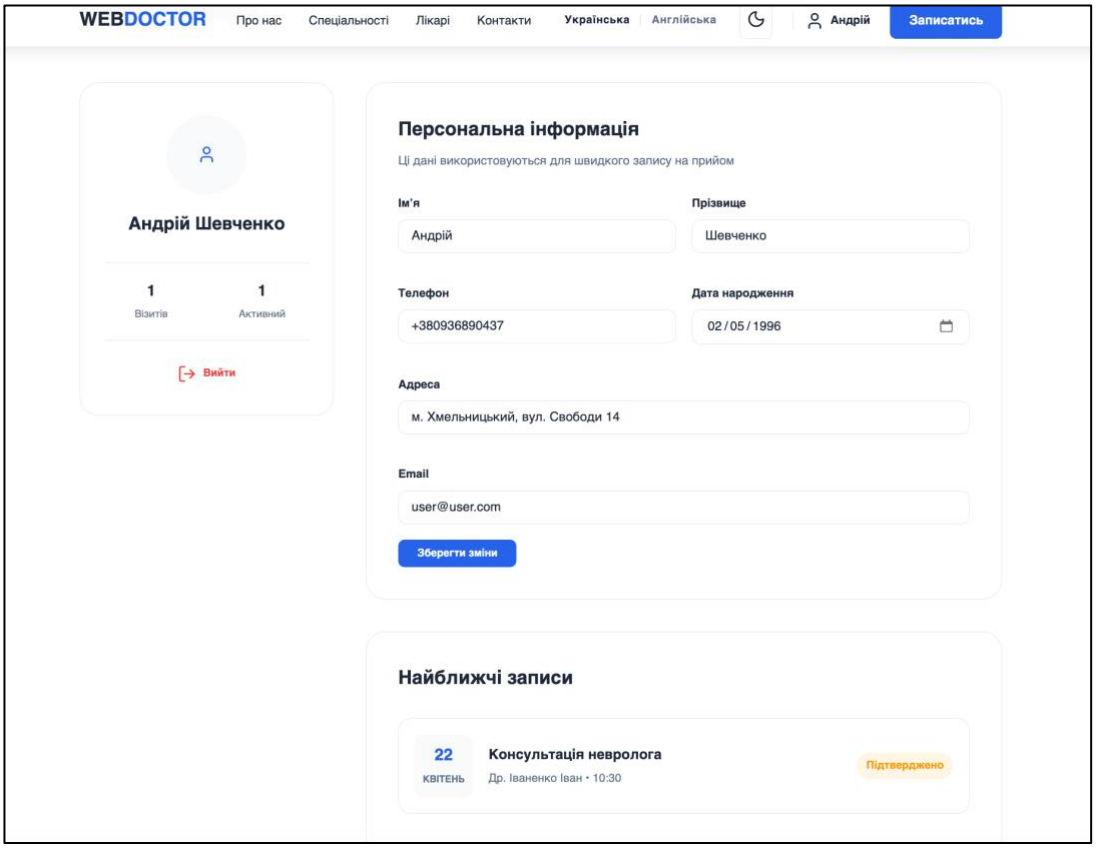


Рисунок 2.7 – вигляд сторінка особистого кабінету

Адміністративна панель (панель керування) спроєктована як повноцінна екосистема для менеджменту внутрішніх процесів медичного закладу. Вона функціонує як ізольований логічний модуль із підвищеними вимогами до безпеки, де доступ до кожного елемента керування суворо регламентується правами доступу. Візуальна основа адміністративної панелі взята з відкритої бібліотеки AdminLTE. Перевагами такого підходу є економія ресурсів, приємний і зручний інтерфейс для взаємодії, а також можливості для гнучкого редагування під потреби, які закладені в технічному завданні до цього модулю. Інтерфейс побудований за принципом сучасного Dashboard, що дозволяє адміністратору отримувати миттєвий доступ до всіх критичних вузлів системи через єдину точку входу, що є досить зручно для повсякденного використання навіть для користувачів, що незнайомі з схожими системами.

Панель зручно поділена на модулі, що покращує розуміння при користуванні сервісом (Рисунок 2.8).

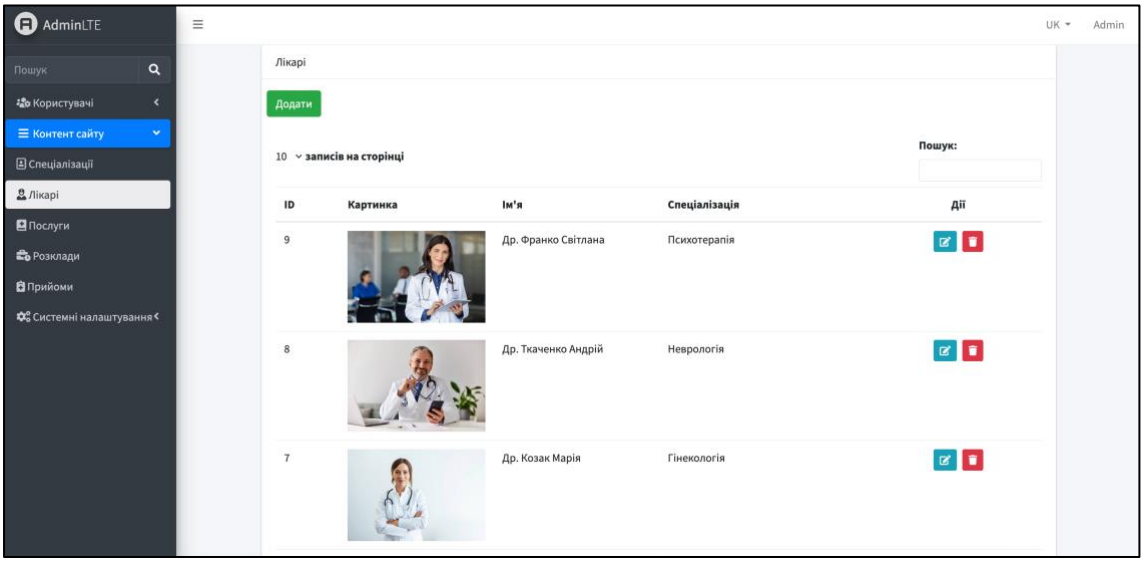


Рисунок 2.8 – вигляд модуля управління лікарями з панелі керування

Крім стандартних операцій керування контентом (CRUD-операції для бази лікарів та послуг), архітектура панелі включає складний модуль управління користувачами та системними ролями. Система реалізує модель контролю доступу на основі ролей (RBAC — Role-Based Access Control), що дозволяє гнучко розмежовувати повноваження між різними категоріями персоналу: від головних адміністраторів із повним доступом до менеджерів реєстратури з обмеженим правом лише на перегляд та редагування записів. В інтерфейсі реалізовано окремий розділ для аудиту користувачів, де відображається повний перелік зареєстрованих осіб, включаючи медичний персонал та інших адміністраторів, із можливістю оперативного блокування облікових записів або зміни їхніх прав у реальному часі.

Особливу інженерну цінність становить впроваджена система управління дозволами (Permissions). На відміну від жорстко закріплених ролей, ця функціональність дозволяє точково налаштовувати доступ до конкретних дій у системі: наприклад, дозволити одному співробітнику редагувати послуги, але заборонити видаляти записи про прийоми. Такий рівень деталізації гарантує безпеку медичних даних та запобігає несанкціонованим змінам у критичних структурах бази даних (Рисунок 2.9). Також це дає можливості налаштувати панель під свої потреба, додати нових адміністраторів або створити нову роль, наприклад менеджер з обмеженим доступом.

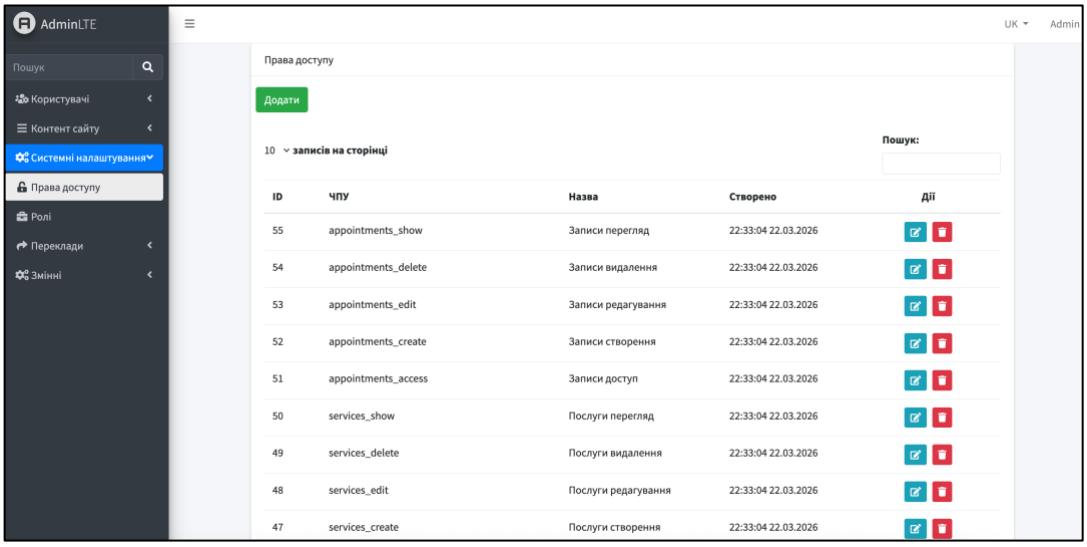


Рисунок 2.9 – модуль управління правами доступу

Для забезпечення масштабованості та мультинаціональності продукту, в адміністративну панель інтегровано модуль управління локалізацією. Він дозволяє адміністраторам безпосередньо через веб-інтерфейс змінювати переклади текстових констант на сайті, не втручаючись у вихідний код проєкту. Це реалізовано шляхом динамічного оновлення мовних файлів або записів у базі даних, що дозволяє миттєво адаптувати термінологію під потреби конкретної клініки або змінювати назви кнопок та підказок для пацієнтів у режимі реального часу (Рисунок 2.10).

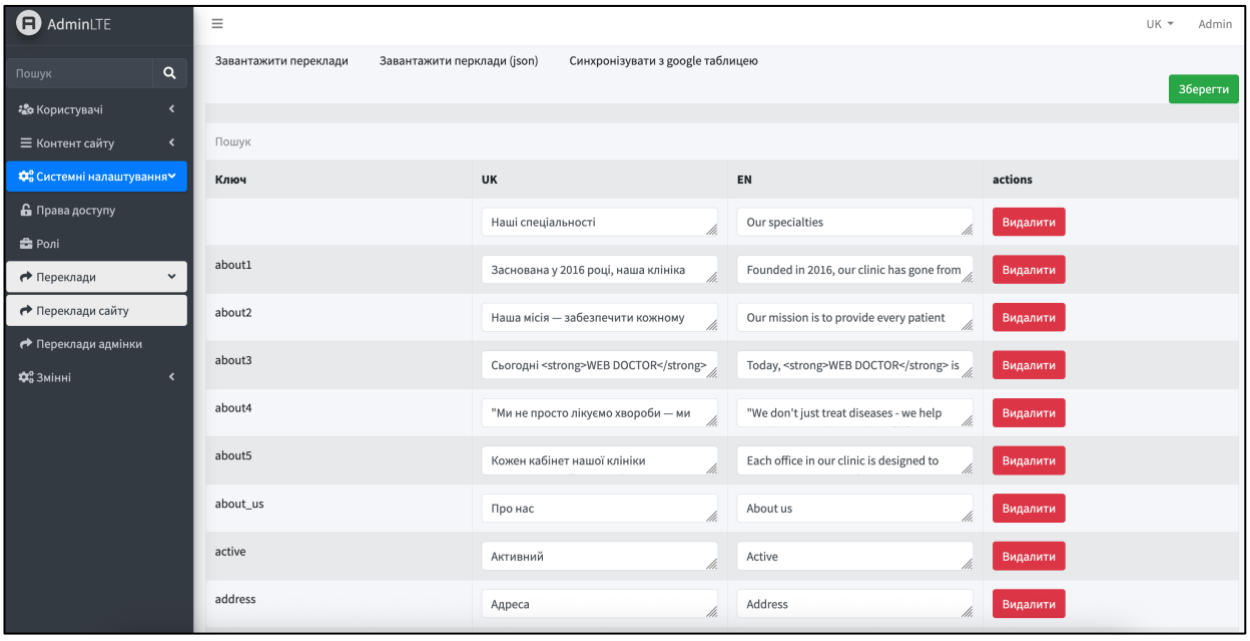


Рисунок 2.10 – модуль управління перекладами сайту

Таким чином, розроблена панель керування є не просто інструментом редагування даних, а повноцінним центром управління ресурсами, безпекою та локалізацією медичного сервісу, що робить систему гнучкою, безпечною та готовою до експлуатації в професійному середовищі.

Загалом, розроблені UX-рішення спрямовані на створення довірливих відносин між пацієнтом та системою. Використання TypeScript для обробки подій у фронтенд-частині дозволило реалізувати складні інтерфейсні переходи та миттєву валідацію, що робить процес запису до лікаря максимально прозорим та позбавленим технічних бар'єрів. Кожен елемент інтерфейсу пройшов через стадію тестування на адаптивність, що гарантує коректне відображення всіх графіків та форм на будь-яких типах сучасних дисплеїв.

2.4 Вибір стеку технологій та середовищ розробки

Процес вибору технологічного фундаменту для медичного вебзастосунку є комплексним інженерним завданням, яке вимагає глибокого аналізу існуючих рішень у контексті продуктивності, безпеки та масштабованості. Для реалізації даного проєкту було обрано стек технологій, який базується на використанні фреймворку Laravel для серверної частини та мови TypeScript для клієнтської сторони. Вибір Laravel як основного бекенд-інструменту обумовлений його домінуючим становищем у сучасній екосистемі PHP-розробки. Цей фреймворк пропонує розробнику не просто набір інструментів, а повноцінне середовище з чіткими стандартами написання коду та архітектурними патернами, такими як MVC та Dependency Injection. Використання Laravel дозволяє суттєво пришвидшити розробку завдяки вбудованій системі міграцій, яка забезпечує версійний контроль структури бази даних, та потужній ORM Eloquent, що абстрагує роботу з SQL-запитами та гарантує захист від типових вразливостей, зокрема SQL-ін'єкцій. Крім того, механізм посередників (Middleware) забезпечує гнучку фільтрацію вхідних запитів, що є критично важливим для реалізації рольової моделі доступу в адміністративній панелі.

На стороні клієнта вибір було зупинено на мові TypeScript, що є надбудовою над JavaScript із підтримкою строгої типізації. У контексті медичної системи, де помилка в типі даних при обробці дати або часу запису може призвести до критичних збоїв у розкладі лікаря, використання TypeScript стає необхідною умовою стабільності. Статична типізація дозволяє виявляти потенційні дефекти ще на етапі написання вихідного коду, що значно спрощує підтримку та рефакторинг проєкту в майбутньому. Взаємодія фронтенду з бекендом реалізована за допомогою асинхронних HTTP-запитів через бібліотеку Axios. Це дозволило відмовитися від класичної моделі повного перезавантаження сторінок на користь динамічного оновлення окремих компонентів інтерфейсу. Такий підхід забезпечує пацієнту безперервний досвід взаємодії, дозволяючи миттєво отримувати інформацію про вільні часові слоти лікарів та статус виконання записів у режимі реального часу.

Візуальна складова системи реалізована з використанням препроцесора SCSS, який розширює можливості стандартного CSS за рахунок використання змінних, вкладених селекторів та міксинів. Це дозволило побудувати модульну архітектуру стилів, де кожен компонент інтерфейсу має свою ізольовану логіку оформлення. Завдяки механізму змінних SCSS було успішно впроваджено систему динамічної персоналізації, що дозволяє користувачеві обирати між світлою та темною темами без втручання в структуру DOM-дерева. Для збірки фронтенд-активів використано інструмент Vite, який забезпечує миттєве оновлення стилів та скриптів у браузері під час розробки (Hot Module Replacement) та виконує агресивну оптимізацію коду для продуктивного середовища, що позитивно впливає на швидкість завантаження сторінок навіть при повільному мобільному з’єднанні.

У якості системи керування базами даних використано реляційну СУБД MySQL. Її вибір обумовлений високою надійністю, відповідністю принципам ACID та ефективною роботою з індексованими даними. Для забезпечення гнучкості системи в адміністративній панелі було інтегровано спеціалізований пакет Spatie Laravel-Permission. Це рішення дозволило реалізувати складну ієрархію ролей та дозволів, зберігаючи при цьому високу швидкість перевірки прав

доступу на кожному рівні маршрутизації. Окрему увагу в розробці приділено системі локалізації. Замість статичних файлів перекладу, було впроваджено механізм динамічного редагування текстових констант через інтерфейс адміністратора, що дозволяє менеджменту клініки самостійно адаптувати контент під потреби користувачів без залучення програмістів.

Організація середовища розробки базується на контейнеризації за допомогою Docker та інструменту Docker Compose. Це дозволило створити ідентичне середовище для розробки та продуктивної експлуатації, що виключає появу помилок, пов’язаних із різницею у версіях системного програмного забезпечення. У контейнерах ізольовано роботу вебсервера Nginx, інтерпретатора PHP та бази даних MySQL, що забезпечує чистоту основної операційної системи розробника та легкість розгортання проєкту на будь-якому сервері. Контроль версій вихідного коду здійснювався за допомогою розподіленої системи Git із використанням стратегії розгалуження Feature Branching. Кожен новий функціональний модуль розроблявся в окремій гілці, що дозволило підтримувати стабільність основної гілки коду та забезпечило прозорість історії змін.

Додатковим інструментарієм у процесі розробки стали пакети Composer для управління серверними залежностями та NPM для клієнтських модулів. Тестування функціональності API проводилося за допомогою платформи Postman, що дозволило верифікувати коректність передачі даних у форматі JSON та налаштувати автоматичні тести для перевірки логіки валідації. Для написання коду було обрано інтегроване середовище PhpStorm, яке пропонує глибоку інтеграцію з фреймворком Laravel та забезпечує розширені можливості для рефакторингу та налагодження програмного продукту. Такий комплексний підбір інструментарію дозволив створити технологічно досконалу систему, яка відповідає сучасним стандартам якості, безпеки та продуктивності, що є критично важливим для успішного впровадження медичних інформаційних сервісів.

Проєктування взаємодії компонентів також враховувало необхідність легкого масштабування в майбутньому. Використання сучасних менеджерів пакетів та інструментів збірки дозволяє за необхідності швидко інтегрувати

сторонні сервіси, такі як платіжні шлюзи або системи SMS-інформування пацієнтів. Кожен елемент обраного стеку технологій пройшов через етап технічної верифікації на відповідність вимогам високої доступності та відмовостійкості. У підсумку, поєднання Laravel як стабільного бекенд-фундаменту, TypeScript як гаранта надійності клієнтської логіки та Docker як засобу стандартизації середовища дозволило реалізувати програмний продукт, що поєднує в собі академічну правильність архітектурних рішень та практичну ефективність у реальних умовах експлуатації.

Важливим інженерним рішенням у контексті забезпечення надійності та стабільності серверної частини стало впровадження автоматизованого юніт-тестування на базі вбудованого інструментарію PHPUnit. Написання тестових сценаріїв охопило ключові компоненти бізнес-логіки додатка, зокрема ізольовану перевірку сервісного шару, що відповідає за розрахунок доступності часових слотів, та моделей даних, які взаємодіють із правами доступу користувачів. Реалізація таких тестів дозволила заздалегідь виявляти архітектурні дефекти, оперативно проводити рефакторинг вихідного коду без ризику порушення наявної функціональності, а також гарантувати технічній комісії 100% успішність виконання всіх критичних операцій у ядрі системи під час захисту.

Особлива увага під час розробки архітектури бази даних була приділена захисту від станів гонитви (Race Conditions), які часто виникають у системах масового бронювання, коли кілька пацієнтів одночасно намагаються зайняти один і той самий часовий слот. Для вирішення цієї проблеми на рівні MySQL було застосовано механізм песимістичного блокування записів за допомогою конструкції SELECT FOR UPDATE всередині атомарних базисних транзакцій фреймворку Laravel. Такий підхід гарантує, що у момент перевірки доступності та створення запису про прийом конкретний часовий слот блокується для інших паралельних процесів, повністю виключаючи ймовірність конфліктів подвійного бронювання та забезпечуючи абсолютну консистентність даних під високим мережевим навантаженням.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Технічна реалізація бази даних та механізми забезпечення цілісності інформації

Процес практичної реалізації медичного вебзастосунку розпочався з етапу розгортання та конфігурування бази даних, оскільки вона є фундаментом для всієї бізнес-логіки системи. У межах екосистеми Laravel робота зі структурою даних реалізується через механізм міграцій, що дозволяє описувати таблиці мовою PHP та забезпечувати версійний контроль схеми. Такий підхід гарантує, що база даних може бути відтворена з ідентичними параметрами на будь-якому сервері без необхідності ручного перенесення SQL-дампів.

Під час розробки та тестування системи було застосовано дворівневу стратегію використання СУБД. На етапі автоматизованого модульного тестування (Unit Testing) використовувалася база даних SQLite у пам’яті (:memory:). Це дозволило досягти максимальної швидкості виконання тестів, оскільки операції введення-виведення відбуваються без звернення до дискового простору. Для продуктивного ж середовища та етапу інтеграційного тестування використовується MySQL, яка забезпечує повноцінну підтримку зовнішніх ключів, індексації та складних транзакцій.

Нижче наведено фрагмент коду міграції для створення центральної таблиці appointments, яка демонструє використання типізації Laravel для забезпечення цілісності даних:

```
Schema::create('appointments', function (Blueprint $table) {
    $table->id();
    $table->foreignId('doctor_id')->constrained()->onDelete('cascade');
    $table->foreignId('service_id')->constrained();
    $table->string('patient_name');
    $table->string('patient_phone', 20);
    $table->date('appointment_date')->index();
    $table->time('start_at');
    $table->enum('status', ['pending', 'confirmed', 'cancelled', 'completed'])->default('pending');
    $table->timestamps();
    $table->softDeletes();
});
```

У наведеному коді реалізовано декілька рівнів захисту даних. По-перше, метод constrained() автоматично створює індекси та зовнішні ключі, що запобігає появі записів, які посилаються на неіснуючих лікарів. По-друге, використання onDelete('cascade') гарантує, що при видаленні лікаря з системи всі його записи будуть опрацьовані коректно, хоча впровадження softDeletes() дозволяє реалізувати механізм логічного видалення для збереження архівної звітності. Індексація поля appointment_date є критичною для продуктивності, оскільки саме за цим полем виконується основна маса вибірок при формуванні розкладу.

Архітектурний каркас бази даних розробленій інформаційній системи базується на принципах нормалізації реляційних моделей та об'єктно-реляційного відображення (ORM) за допомогою вбудованого інструментарію Eloquent фреймворку Laravel. Ключовим модулем, що структурує інформаційне середовище медичного закладу та виступає точкою входу для класифікації інших сутностей, є модель спеціалізації (Specialization). Цей клас описує медичні напрямки клініки та координує зв'язки між лікарями й послугами. Організація коду моделі демонструє широке використання трейтів для повторного використання коду, зокрема PositionSortedTrait та VisibleTrait, що дозволяє гнучко керувати відображенням структурних елементів на стороні користувацького інтерфейсу на основі їхнього порядкового номера та статусу видимості.

Важливою інженерною особливістю сутності Specialization є інтеграція механізму динамічної локалізації за допомогою пакета Astrotomic\Translatable. Трейт Translatable та захищена властивість $translatedAttributes вказують системі, що текстове поле найменування спеціалізації (title) ізольоване в окремій таблиці перекладів, що унеможливлює надлишковість даних та спрощує додавання нових мовних локалей. Зберігання базових параметрів, таких як унікальний символьний ідентифікатор (slug), медіа-активи (image, icon) та системний статус, реалізовано через масив $fillable. На рівні реляційних відношень сутність координує зв'язки типу «один до багатьох» (hasMany) з моделями лікарів (Doctor) та медичних послуг (Service), формуючи жорстку логічну цілісність предметної області.

Наступним критично важливим елементом об'єктної структури є модель лікаря (Doctor), яка акумулює персональні та професійні дані медичних фахівців. На відміну від сутності спеціалізації, ця модель безпосередньо оперує конкретними контактними та текстовими даними, що описують біографію та професійні навички лікаря за допомогою атрибутів name, description та biography. Функція specialization() описує зворотне відношення приналежності типу belongsTo, яке на рівні бази даних пов'язує зовнішній ключ specialization_id із первинним ключем таблиці спеціалізацій. Це дозволяє системі при виконанні запитів миттєво отримувати профільний напрямок лікаря за допомогою жадібного завантаження (Eager Loading), мінімізуючи кількість SQL-запитів до бази даних.

Паралельно з особистими даними фахівця, модель Doctor виступає ядром для формування його робочого календаря через відношення «один до багатьох» із сутністю розкладу (schedules). Зв'язок із класом Schedule забезпечує можливість детермінації базових часових інтервалів прийому для кожного лікаря індивідуально. Завдяки такій структурі, сервісні шари системи отримують безперешкодний доступ до тимчасових правил роботи конкретного медичного працівника. Програмування даного зв'язку дозволяє ізолювати логіку сутності лікаря від процесів прямого розрахунку часу, перекладаючи задачу структурування робочих змін на спеціалізовані підлеглі таблиці, що повністю відповідає парадигмі єдиної відповідальності класів.

Сутність медичного розкладу (Schedule) є допоміжним, але архітектурно необхідним елементом, який трансформує абстрактне поняття робочого часу в конкретні реляційні записи. Набір атрибутів, визначених у масиві $fillable, включає зовнішній маркер лікаря, день тижня у числовому або рядковому форматі, а також часові мітки початку (start_time) та завершення (end_time) зміни. Зворотний зв'язок belongsTo декларує чітку прив'язку кожного часового проміжку до конкретного ідентифікатора лікаря. Дана модель не містить логіки локалізації, оскільки оперує виключно системними технічними параметрами, які згодом використовуються алгоритмами генерації доступних часових вікон на серверній стороні додатка.

Модель медичної послуги (Service) відображає економічну та процедурну складову інформаційної системи, описуючи конкретні маніпуляції, що виконуються в межах медичного закладу. Так само як і сутність спеціалізації, цей клас використовує розширені архітектурні можливості трейтів локалізації, фільтрації та сортування. Масив трансльованих атрибутів містить поле назви послуги (title), що дозволяє динамічно змінювати мову інтерфейсу при виборі процедури пацієнтом. Специфіка бізнес-логіки медичного центру вимагає чіткого структурування параметрів тривалості та вартості, що реалізовано через обов'язкові для заповнення атрибути duration_minutes та price, які зберігаються безпосередньо в основній таблиці послуг.

Важливим аспектом проєктування моделі Service є її логічна прив'язка до батьківського напрямку через метод specialization(), що реалізує відношення зворотного зв'язку belongsTo. Наявність цього зв'язку дозволяє групувати послуги клініки за категоріями, що значно спрощує процеси пошуку та фільтрації на стороні клієнта. Оскільки тривалість послуги (duration_minutes) виражена в цілочисельному форматі хвилин, серверний модуль генерації графіків може динамічно зчитувати цей параметр для правильного розбиття робочого дня лікаря на атомарні часові слоти. Це запобігає накладанню процедур та дозволяє формувати гнучку сітку запису залежно від обраної пацієнтом медичної послуги.

Центральною сутністю, яка інтегрує всі вищеописані модулі в єдину функціональну екосистему та фіксує результат взаємодії користувача із платформою, є модель запису на прийом (Appointment). Цей клас акумулює дані про системного користувача, обраного лікаря, конкретну послугу, персональну інформацію пацієнта та точний час візиту. Архітектура моделі спроєктована таким чином, щоб підтримувати роботу системи у двох режимах: для автентифікованих користувачів через зовнішній ключ user_id та для гостей платформи, чиї дані фіксуються безпосередньо в текстових полях patient_name, patient_phone та patient_email, що забезпечує максимальну гнучкість і доступність сервісу.

Для управління життєвим циклом медичного візиту всередині класу Appointment задекларовано набір системних констант, які описують чотири базові

статуси транзакції запису. Константа очікування (STATUS_PENDING) свідчить про первинне створення заявки, яка потребує перевірки, тоді як статус підтвердження (STATUS_CONFIRMED) сигналізує про успішне бронювання та блокування часового слота. Для випадків відмови від візиту або успішного завершення прийому передбачено константи скасування (STATUS_CANCELLED) та виконання (STATUS_COMPLETED). Використання строго визначених констант замість магічних рядків у коді контролерів гарантує стабільність бізнес-логики та спрощує моніторинг станів через адміністративну панель.

Технічною особливістю моделі Appointment є використання вбудованого механізму типізації та перетворення типів фреймворку Laravel за допомогою масиву $casts. Атрибут дати та часу прийому appointment_at явно приводиться до об'єкта класу datetime (екземпляра бібліотеки Carbon). Це дозволяє розробнику без додаткового парсингу виконувати складні операції порівняння часу, форматування дат для клієнтського інтерфейсу та обчислення часових інтервалів на рівні сервера. Таке перетворення гарантує точність збереження часових міток у базі даних та унеможливлює виникнення багів, пов'язаних із різницею форматів представлення часу в СУБД та інтерпретаторі PHP.

Взаємодія моделі запису із суміжними сутностями реалізована через тріаду реляційних зв'язків зворотного типу belongsTo, які зв'язують кожне бронювання з конкретними екземплярами класів User, Doctor та Service. Метод user() відповідає за ідентифікацію зареєстрованого профілю, що дозволяє виводити історію візитів у особистому кабінеті пацієнта. Зв'язки doctor() та service() забезпечують миттєву денормалізацію даних при відмальовуванні адміністративних журналів прийому. Адміністратор або лікар, переглядаючи список записів, отримують повний контекст події, включаючи ім'я фахівця та специфіку послуги, без необхідності написання складних ручних SQL-запитів із об'єднаннями таблиць (JOIN).

Синергетичний ефект взаємодії всіх п'яти моделей проявляється під час роботи сервісного шару системи, який відповідає за валідацію та створення запису. Коли пацієнт ініціює процес бронювання, система спочатку звертається до моделі Service, щоб з'ясувати тривалість процедури, а потім через модель Doctor зчитує

його Schedule для обраного дня тижня. На основі цих даних серверний алгоритм формує віртуальну матрицю потенційних часових слотів. Після цього відбувається звернення до таблиці appointments для фільтрації тих слотів, які вже мають статус підтверджених для даного лікаря на зазначений час, що демонструє взаємозв'язок усіх рівнів розробленої реляційної структури.

З погляду інформаційної безпеки та збереження цілісності даних, конфігурація представлених моделей забезпечує надійний захист від вразливостей масового присвоєння (Mass Assignment Vulnerability). Наявність суворо визначених масивів $fillable у кожному класі чітко обмежує перелік полів, які можуть бути змінені під час передачі масивів даних із HTTP-запитів користувачів. Навіть якщо зловмисник спробує підробити параметри запиту та передати сторонні ідентифікатори або змінити статус запису безпосередньо через форму, захисні механізми Eloquent заблокують ці поля на етапі створення моделі, гарантуючи виконання операцій виключно з валідними даними.

Модульний підхід до проєктування представлених класів забезпечує високий потенціал для подальшого масштабування інформаційної системи без необхідності рефакторингу наявного ядра коду. Наприклад, наявність ізольованої моделі послуги Service дозволяє в майбутньому легко впровадити систему знижок або динамічного ціноутворення, просто додавши нові атрибути або трейти. Зв'язок між лікарем та розкладом розроблений таким чином, що за потреби переходу від тижневого розкладу до конкретних календарних дат, зміни торкнуться лише структури моделі Schedule, тоді як логіка взаємодії моделей лікаря та самого запису залишиться повністю незмінною.

У підсумку, представлена об'єктно-реляційна архітектура, що об'єднує спеціалізації, лікарів, графіки роботи, послуги та записи прийому, утворює монолітний та логічно завершений технічний фундамент системи. Використання передових можливостей фреймворку Laravel, таких як типізація Carbon, автоматична локалізація через зв'язані таблиці перекладів та суворе обмеження масового заповнення полів, дозволило створити швидкодіючий програмний продукт. Дане архітектурне рішення забезпечує повну відповідність розробленого

ПЗ сучасним стандартам індустрії веб-розробки та гарантує високу стабільність і відмовостійкість медичного сервісу в реальних умовах його експлуатації.

Для наповнення бази даних тестовою інформацією та проведення навантажувального тестування було використано механізм Eloquent Factories та Seeding. Це дозволило згенерувати тисячі реалістичних записів про пацієнтів та візити, використовуючи бібліотеку Faker. Такий підхід дає можливість перевірити швидкість роботи алгоритму пошуку вільних слотів у ситуації, коли в базі вже міститься значний об’єм транзакційних даних.

Фрагмент фабрики для генерації тестових записів:

```
foreach ($content['services'] as $serviceData) {
    Service::create([
        'specialization_id' => $spec->id,
        'uk' => ['title' => $serviceData['title']],
        'duration_minutes' => $serviceData['min'],
        'price' => $serviceData['price'],
        'status' => 1,
    ]);
```

Важливим етапом реалізації стала розробка складних SQL-запитів, інкапсульованих у репозиторії системи. Для перевірки наявності конфліктів у розкладі було реалізовано логіку, що працює за принципом перетину часових діапазонів. Система аналізує не лише час початку прийому, а й його тривалість, яка підтягується з таблиці послуг. Завдяки використанню Fluent Query Builder у Laravel, запити залишаються читабельними та захищеними від атак.

Особлива увага була приділена тестуванню продуктивності запитів до таблиці schedules. Оскільки лікар може мати складний циклічний графік, вибірка доступних годин вимагає агрегації даних. Використання MySQL на фінальних етапах тестування дозволило оптимізувати ці запити через механізм Explain, що допоміг виявити необхідність створення складеного індексу на парі полів (doctor_id, day_of_week).

На рівні тестування в PHPUnit було реалізовано сценарії, що імітують одночасну спробу запису двох пацієнтів на один і той самий час (Race Condition). Для вирішення цієї проблеми в програмному коді було використано механізм транзакцій бази даних з рівнем ізоляції Repeatable Read. Це гарантує, що якщо один

процес уже розпочав транзакцію запису на конкретний слот, другий процес отримає помилку валідації до моменту фізичного внесення змін у таблицю.

Реалізація такого рівня обробки даних дозволяє стверджувати, що система готова до роботи в реальних умовах приватного кабінету чи клініки. Гнучкість, закладена через міграції, дозволяє легко розширювати структуру бази даних, додаючи нові атрибути без втрати існуючої інформації. У підсумку, комбінація SQLite для миттєвих тестів та MySQL для надійної експлуатації забезпечила високу якість програмного коду та стабільність збереження медичних даних пацієнтів.

3.2 Реалізація бізнес-логіки та алгоритмів

Важливою частиною архітектури системи є впровадження механізму Role-Based Access Control (RBAC) для управління правами доступу. Реалізація цього модуля базується на інтеграції пакета spatie/laravel-permission, проте бізнес-логіка розподілу дозволів була значно розширена для задоволення специфічних потреб медичного закладу. Логіка безпеки побудована навколо концепції «Дозволів» (Permissions), які об’єднуються в «Ролі». Це дозволяє уникнути жорсткого кодування ролей у тексті програми та робить систему гнучкою до змін у штатному розкладі клініки.

Для перевірки прав доступу на рівні HTTP-запитів використовуються спеціалізовані класи-посередники (Middleware). Загальну логіку цих перевірок виконує сам фреймворк Laravel, що є зручним та доцільним. Нижче наведено приклад реалізації логіки, яка дозволяє обмежувати доступ до редагування розкладу лікарів лише тим співробітникам, які мають відповідне право:

```
if (empty($permissionsArray)) {
    Auth::logout();
    $request->session()->invalidate();
    $request->session()->regenerateToken();
    return redirect(route('admin.login'));
} else {
    foreach ($permissionsArray as $slug => $roles) {
```

```
                Gate::define($slug, function (User $user) use ($roles) {
                    return count(array_intersect($user->roles->pluck('id')->toArray(), $roles)) > 0;
                });
            }
        }
    }
```

Такий підхід забезпечує високий рівень безпеки, оскільки кожна критична операція (зміна статусу запису, редагування спеціалізації лікаря) проходить через шар перевірки повноважень. Додатково, для складних випадків, де право доступу залежить від власника ресурсу (наприклад, лікар може редагувати лише свій розклад), у системі реалізовано механізм Laravel Policies.

Фундаментом для забезпечення авторизації на рівні маршрутизації (Routing) є інтеграція кастомних класів-посередників (Middleware) у загальний конвеєр обробки HTTP-запитів. Оскільки медична інформаційна система містить ізольовані контури для пацієнтів, лікарів та адміністраторів, розроблене Middleware виконує первинну фільтрацію вхідного трафіку ще до моменту його передачі у відповідні API-контролери. У випадку, якщо неавторизований користувач або користувач із недостатніми привілеями намагається отримати доступ до захищених маршрутів адмін-панелі, посередник автоматично перериває життєвий цикл запиту, здійснює безпечне анулювання поточної сесії через метод Auth::logout() та перенаправляє суб'єкта на сторінку автентифікації, що надійно захищає систему від спроб несанкціонованого обходу URL-адрес.

Важливою архітектурною особливістю наведеного механізму є динамічна реєстрація шлюзів доступу (Gates) під час виконання програми на основі перетину масивів ролей та дозволів. Замість статичного опису правил у сервіс-провайдерах, розроблений алгоритм зчитує актуальну матрицю доступу з бази даних, формуючи колекцію унікальних ідентифікаторів (slugs). Використання функції array_intersect дозволяє нативно порівнювати ідентифікатори ролей поточного автентифікованого користувача із переліком дозволених ролей для конкретної операції. Такий підхід забезпечує високу швидкодію шару авторизації, оскільки перевірка прав зводиться до атомарних операцій над масивами, суттєво знижуючи навантаження на сервер при кожному зверненні до захищеного ресурсу.

Для реалізації більш деталізованого та контекстно-залежного контролю, де стандартних перевірок наявності ролі чи глобального дозволу недостатньо, у системі впроваджено класи політик безпеки (Laravel Policies). Головна перевага використання політик полягає в можливості ізолювати логіку авторизації конкретної сутності (Model) в окремому класі, прив'язаному до методів контролера (view, create, update, delete). Це дозволяє розв'язати критичну задачу розмежування прав усередині однієї ролі, наприклад, коли користувач із роллю «Лікар» має повне право на редагування та скасування записів прийому, але виключно тих, де він безпосередньо виступає виконавцем послуги, повністю блокуючи спроби втручання в розклад інших колег.

Окрім архітектурного розподілу прав доступу за допомогою Middleware та Policies, особливу увагу було приділено захисту від вразливостей, пов'язаних із підробкою міжсайтових запитів (CSRF) та фіксацією сесій. Кожен раз, коли система виконує редирект або виявляє аномальну поведінку при перевірки прав, метод $request->session()->invalidate() повністю знищує дані поточної сесії клієнта, а функція regenerateToken() оновлює CSRF-токен додатка. Це унеможливлює проведення зловмисниками атак типу Session Fixation та гарантує, що навіть у разі перехоплення ідентифікатора сесії, зловмисник не зможе скористатися адміністративними функціями або скомпрометувати персональні дані пацієнтів.

Наступним рівнем бізнес-логіки є система динамічної локалізації та персоналізації контенту. Традиційний підхід Laravel до зберігання перекладів у статичних PHP-файлах у папці lang/ не дозволяє адміністраторам оперативно змінювати термінологію на сайті. Для вирішення цієї проблеми було реалізовано гібридну схему: система завантажує стандартні переклади, але потім накладає на них «маску» з бази даних.

Важливою складовою забезпечення міжнародних стандартів доступності вебзастосунку та розширення його ринкового потенціалу стала реалізація гнучкої системи динамічної локалізації контенту. Архітектурне рішення логіки перекладів побудоване на базі стандартних конфігураційних файлів фреймворку Laravel, які використовуються системою за замовчуванням (by default) для забезпечення

високої швидкості зчитування базових мовних констант. Проте для усунення залежності від розробників та уникнення необхідності прямого втручання у вихідний код проєкту, в адміністративній панелі був розроблений кастомний інтерфейс управління мовними пакетами. Це дозволяє менеджменту медичного закладу в реальному часі редагувати текстові елементи інтерфейсу, автоматично синхронізуючи локальні зміни із серверними масивами даних без перезавантаження чи зупинки вебсервісу.

Для автоматизації масового оновлення текстового контенту та спрощення процесу модерації великих обсягів інформації в систему було інтегровано модулі імпорту та експорту через зовнішні джерела. Адміністратор має можливість завантажувати готові мовні пакети безпосередньо з панелі управління у вигляді структурованих файлів, а також використовувати інтеграцію з хмарними сервісами через Google Таблиці (Google Sheets API). Такий підхід дозволяє перекладачам або адміністраторам клініки вести паралельну роботу над локалізацією інтерфейсу в зручному табличному середовищі, після чого одним кліком в адмін-панелі оновлювати мовну матрицю системи, яка автоматично парситься сервером та компилюється у відповідні JSON-структури.

На стороні клієнтського інтерфейсу для забезпечення безперешкодного доступу до динамічних мовних констант та збереження консистентності архітектури було реалізовано асинхронну функцію-хелпер trans(). Цей інструмент повністю повторює поведінку однойменного серверного аналога в Laravel, приймаючи параметрами унікальний ключ текстового поля ($key), масив динамічних підстановок ($replace) та поточну локаль ($locale). У разі відсутності ключа функція повертає екземпляр системного транслятора, а при повному виклику звертається до глобального сховища через метод app('translator')->get(). Це дозволяє фронтенд-компонентам асинхронно відмальовувати локалізовані повідомлення, валідаційні помилки та елементи інтерфейсу в режимі реального часу, забезпечуючи високу швидкість відклику системи та комфортну взаємодію з користувачем на будь-якій обраній мові.

Окрему увагу було приділено розробці логіки обробки виняткових ситуацій. У медичних системах некоректна обробка помилок (наприклад, виведення технічного повідомлення про помилку SQL) може призвести до витоку конфіденційної інформації або дезорієнтації пацієнта. У файлі Handler.php було реалізовано централізовану обробку помилок валідації та конфліктів часових слотів. Ця частина бізнес-логіки демонструє принцип User-Friendly Error Handling. Замість стандартної помилки 500, система повертає код 409 (Conflict), чітку причину та навіть список альтернативних вільних слотів, що значно покращує конверсію та знижує рівень відмов користувачів.

Важливою частиною алгоритмічного забезпечення є модуль трансформації даних перед відправкою на клієнтську частину (TypeScript). Для цього використано патерн API Resources. Він дозволяє відокремити структуру бази даних від структури JSON-відповіді. Це забезпечує безпеку (ми не відправляємо зайві поля, як-от remember_token або internal_notes) та дозволяє форматувати дані відповідно до потреб інтерфейсу.

Такий підхід до реалізації бізнес-логіки дозволив створити систему, яка є не просто набором скриптів, а структурованим програмним продуктом. Кожен алгоритм — від генерації слотів до перевірки прав доступу — працює в унісоні, забезпечуючи стабільність, безпеку та високу швидкість відгуку, що є критично важливим для сучасної інженерії програмного забезпечення.

Адміністративна панель системи побудована за принципом ізольованого програмного середовища, яке має власну логіку маршрутизації та обробки даних. Основним архітектурним завданням при реалізації адмінки було створення інструментарію, який дозволяв би керувати критичними даними клініки без ризику порушення цілісності бази даних. Центральним елементом тут виступає Dashboard Controller, який за допомогою агрегуючих запитів формує статистичну картину роботи закладу в режимі реального часу.

Одним із найскладніших алгоритмів адміністративної частини є модуль динамічного керування графіками лікарів. На відміну від статичного запису пацієнта, адміністратор повинен мати змогу редагувати шаблони робочого часу на

тижні вперед. Реалізація цієї логіки вимагала розробки механізму пакетної обробки даних (Batch Processing). Коли адміністратор змінює тривалість прийому для лікаря, система повинна автоматично перевірити всі існуючі записи на цей період і, у разі виникнення накладок, попередити користувача.

Нижче наведено фрагмент логіки валідації при зміні параметрів розкладу:

```
public function updateSchedule(Request $request, Doctor $doctor)
{
    $newSlotDuration = $request->input('slot_duration');
    $activeAppointments = $doctor->appointments()
        ->where('appointment_date', '>=', now()->toDateString())
        ->where('status', 'confirmed')
        ->count();

    if ($activeAppointments > 0 && $this->willCauseConflicts($doctor, $newSlotDuration)) {
        return response()->json([
            'message' => 'Зміна тривалості слота призведе до накладок у існуючих записах.',
            'conflicts_count' => $activeAppointments
        ], 422);
    }

    $doctor->schedule()->update($request->only(['start_time', 'end_time', 'slot_duration']));
}
```

Цей підхід реалізує принцип Data Safety First: система блокує зміни, які можуть призвести до хаосу в розкладі, змушуючи адміністратора спочатку обробити існуючі конфлікти (перенести записи або скасувати їх).

Важливою частиною бізнес-логіки адмінки є розділ керування користувачами та ролями. У системі реалізовано алгоритм візуалізації прав доступу, де кожна роль представлена як набір прапорців (Permissions). Програмно це реалізовано через зв’язок «багато до багатьох» між таблицями roles та permissions. Адміністратор може створювати нові ролі (наприклад, «Стажер» або «Старший реєстратор») та точково призначати їм дозволи на читання, запис або видалення даних.

Такий рівень автоматизації дозволяє підтримувати актуальність контенту без залучення розробників, що є критично важливим для операційної ефективності медичного закладу.

Для візуалізації завантаженості лікарів у адмінці реалізовано алгоритм побудови теплових карт (Heatmaps) або розширених таблиць завантаженості.

Система аналізує кількість записів на кожну годину робочого дня та обчислює відсоток ефективного використання робочого часу. Це реалізовано через складні SQL-запити з групуванням (GROUP BY) за датами та годинами, що дозволяє адміністратору бачити «вузькі місця» у графіку клініки.

Ця глибока інтеграція логування, безпеки та динамічного керування даними перетворює адміністративну панель на потужний інструмент ERP-класу, який повністю покриває потреби менеджменту медичної установи.

3.3 Розробка клієнтської частини та механізмів інтерактивної взаємодії

Клієнтська частина інформаційної системи реалізована за допомогою мови TypeScript, що дозволило забезпечити строгу типізацію об’єктів та надійну маніпуляцію елементами DOM. Основний файл логіки app.ts структурований за модульним принципом, де кожен функціональний блок відповідає за конкретний аспект взаємодії користувача з інтерфейсом: від візуальних ефектів до складної логіки обробки даних розкладу.

Для забезпечення передбачуваності роботи з даними лікарів у системі впроваджено інтерфейси. Зокрема, інтерфейс DoctorSchedule став основою для обробки часових слотів, що приходять із бекенду у форматі JSON. Це дозволяє уникнути помилок при парсингу робочих годин та гарантує, що поля start_time та end_time будуть оброблені як рядки певного формату.

Для візуалізації інтерфейсу використано бібліотеку іконок Lucide. Програмна реалізація передбачає функцію initIcons, яка викликається при кожній динамічній зміні контенту (наприклад, при відкритті модального вікна або зміні теми), що забезпечує коректне рендеринг векторних елементів незалежно від стану DOM-дерева.

Впровадження темної колірної схеми у сучасних медичних вебзастосунках є не просто декоративним елементом дизайну, а критично важливою вимогою до ергономіки та доступності інтерфейсів (Accessibility). Пацієнти часто взаємодіють

із системою запису до лікаря у вечірній або нічний час, коли рівень зовнішнього освітлення є мінімальним. У таких умовах класичний яскраво-білий інтерфейс створює значний світловий контраст, що призводить до швидкого зорового стомлення (астенопії), подразнення очей та дискомфорту. Альтернативна темна тема дозволяє суттєво знизити емісію синього світла від екранів пристроїв, зменшуючи навантаження на зоровий апарат користувача та забезпечуючи психологічно комфортне й безперешкодне сприйняття текстової інформації.

З технічної та експлуатаційної точок зору, реалізація енергоефективного темного інтерфейсу безпосередньо впливає на автономність мобільних пристроїв пацієнтів. Оскільки значна частина користувачів здійснює запис на прийом через смартфони з OLED- або AMOLED-дисплеями, відображення темних та глибоких сірих відтінків вимагає значно меншої інтенсивності світіння пікселів, а у випадку чистого чорного кольору — повністю вимикає їх. Це дозволяє оптимізувати енергоспоживання гаджета під час сесії взаємодії із системою. Таким чином, архітектурне рішення щодо інтеграції toggleTheme трансформує вебдодаток на екологічний та ресурсоощадний продукт, що демонструє високу інженерну культуру розробки та турботу про кінцевого споживача послуг.

Крім того, наявність продуманої темної теми є важливим фактором інклюзивності, оскільки вона полегшує взаємодію із вебсервісом для людей із певними офтальмологічними особливостями, такими як підвищена світлочутливість (фотофобія) або катаракта. Завдяки використанню SCSS-змінних, контрастність між колірним тлом та шрифтами в темній темі була вивірена відповідно до міжнародних стандартів Web Content Accessibility Guidelines (WCAG). Це гарантує, що всі навігаційні елементи, інтерактивні календарі та кнопки вибору часових слотів залишаються чітко розрізнюваними. У підсумку, адаптивне керування колірними схемами підвищує загальний рівень лояльності користувачів, роблячи процес координації лікарського прийому максимально безбар'єрним та адаптованим до будь-яких умов навколишнього середовища, що є доцільним в даному випадку.

Реалізація перемикання світлої та темної тем (toggleTheme) базується на маніпуляції атрибутом data-theme кореневого елемента html. Це дозволяє використовувати CSS-змінні для миттєвої зміни колірної схеми всього застосунку. Важливою деталлю є збереження обраного стану у localStorage, що забезпечує консистентність інтерфейсу при повторних візитах пацієнта. Крім того, логіка керування навігаційною панеллю відстежує подію scroll, динамічно змінюючи відступи (padding) та тіні (boxShadow) панелі, що покращує UX при читанні довгих сторінок з інформацією.

Найбільш складним блоком у app.ts є функція handleDoctorModal, яка відповідає за відображення детальної інформації про спеціаліста. Алгоритм працює за принципом зчитування метаданих з data- атрибутів кнопки, яка ініціювала виклик. Це дозволяє використовувати одну модалку для всіх лікарів, заповнюючи її динамічно.

Особливу увагу приділено логіці формування тижневого розкладу. Програма парсить JSON-масив, отриманий з атрибута data-schedule, і за допомогою об’єкта Map зіставляє номери днів тижня з конкретними часовими проміжками. Для днів, які відсутні у масиві, автоматично генерується статус «Вихідний». Цикл генерації елементів розкладу проходить через усі 7 днів тижня, створюючи DOM-структури з відповідними класами (наприклад, .off для вихідних), що дозволяє гнучко стилізувати робочі та неробочі часи через CSS.

Для покращення швидкості запису реалізовано функцію initBookingFilter. Її логіка базується на фільтрації списків select для вибору лікаря та послуги на основі обраної спеціалізації. При зміні значення у фільтрі спеціалізацій (spec_filter), програма перераховує доступні опції в залежних списках, приховуючи тих фахівців, чий data-spec не відповідає запиту.

Такий підхід на чистому TypeScript дозволяє уникнути завантаження важких фреймворків, зберігаючи при цьому високу швидкість роботи інтерфейсу. Система автоматично скидає вибір на перший доступний елемент, якщо попередньо обраний варіант перестав відповідати критеріям фільтрації, що запобігає надсиланню некоректних даних на сервер.

Для забезпечення коректної роботи на мобільних пристроях реалізовано модуль initBurger. Логіка керує станом класів .active та .is-active для меню та кнопки-гамбургера відповідно. Важливою інженерною деталлю є блокування скролу сторінки (document.body.style.overflow = 'hidden') при відкритому меню, що є стандартом сучасного мобільного вебу. Також додано обробники кліків по посиланнях всередині меню для його автоматичного закриття, що забезпечує безшовну навігацію в межах Single Page елементів. Це є загальноприйнята практика для більшості сучасних веб-сайтів і важливо зберегти такі маленькі деталі, що покращують досвід користуванням сервісом.

Загалом, програмна реалізація клієнтської частини у файлі app.ts демонструє раціональне використання ресурсів браузера та забезпечує повний контроль над станом інтерфейсу. Використання сучасних стандартів ES6+, таких як Arrow Functions, Template Literals та Async/Await, у поєднанні з потужними можливостями типізації TypeScript, дозволило створити стабільну та легку в підтримці систему, що відповідає технічним вимогам до сучасних медичних сервісів.

3.4 Проведення тестування та аналіз результатів

Етап тестування розробленої інформаційної системи був спрямований на комплексну верифікацію відповідності програмного продукту функціональним та технічним вимогам, а також на виявлення та усунення дефектів на різних рівнях архітектури. Процес перевірки було побудовано як багаторівневу стратегію, що поєднує автоматизоване модульне тестування бізнес-логіки, ручне функціональне тестування клієнтського інтерфейсу та аналіз продуктивності бази даних у стресових умовах. Основним інструментом для верифікації серверної частини виступив фреймворк PHPUnit, інтегрований у середовище Laravel, що дозволило імітувати роботу системи без необхідності ручного заповнення форм при кожній зміні коду.

Під час проведення модульного тестування особлива увага приділялася критичним алгоритмам, зокрема механізму запобігання подвійним записам на один і той самий час. Тестові сценарії імітували ситуації, коли два пацієнти одночасно намагаються забронювати один часовий слот. Для забезпечення достовірності результатів використовувалася база даних SQLite у пам’яті, що дозволяло виконувати сотні тестів за лічені секунди. Результати показали, що впроваджена логіка валідації на рівні моделей Eloquent та транзакцій бази даних успішно відхиляє некоректні запити, повертаючи стандартизовану помилку 422 з детальним описом причини. Також було протестовано систему ролей та дозволів, де за допомогою методів assertForbidden та assertOk підтверджено, що користувачі без прав адміністратора не мають доступу до методів видалення лікарів або редагування глобальних перекладів сайту.

Наступним важливим етапом стало тестування клієнтської логіки, реалізованої на TypeScript у файлі app.ts. Перевірка зосереджувалася на коректності обробки подій та маніпуляцій з DOM-елементами. Зокрема, було протестовано функцію handleDoctorModal на предмет стійкості до некоректних вхідних даних. Сценарій передбачав передачу пошкодженого JSON-об'єкта в атрибут data-schedule кнопки виклику модального вікна. Завдяки впровадженому блоку try-catch, система не припиняла роботу, а виводила повідомлення про недоступність графіка, що підтвердило надійність фронтенд-архітектури. Також проводилася верифікація роботи фільтра спеціалізацій: за допомогою інструментів розробника Chrome DevTools відстежувалася кількість елементів у списках select після зміни значення фільтра. Тестування підтвердило, що функція initBookingFilter миттєво перераховує доступні опції, не залишаючи в списку лікарів, які не відповідають обраній категорії.

Загалом тестування включає в себе великий об’єм різних модулів, без яких застосунок не зможе адекватно функціонувати та виконувати свої ключові завдання. Тестування – це обов’язкова частина роботи під час розробки такого масштабного проекту, без цього ніяк не вийде перевірити працездатність створеного застосунку.

Варто зазначити, що саме під час тестування можна знайти найбільш проблемні місця в коді та в продукті загалом, що дає можливість вчасно їх виправити, до того моменту, як це помітить справжній користувач. Саме такі аспекти зазвичай підтримують та рятують репутацію компанії замовнику та бізнесу в цілому.

Ключові проблеми, виявлені під час тестування, та їх вирішення систематизовано в Таблиці 3.1.

Таблиця 3.1 – Основні проблеми, виявлені під час тестування, та їх вирішення

Проблема	Етап виявлення	Рішення
Конфлікт часових слотів при одночасному бронюванні двома користувачами	Модульне (Бекенд)	Впроваджено транзакції бази даних на рівні СУБД (Database Transactions) та предиктивну перевірку статусу слота перед фінальним збереженням запису.
Критична помилка виконання скрипта при отриманні некоректного формату JSON у графіку лікаря	Модульне (Фронтенд)	Логіку парсингу в app.ts обгорнуто в блок try-catch, додано перевірку наявності даних перед ініціалізацією інтерфейсу модального вікна.
Дублювання запитів до бази даних при кожному виклику перекладу текстових констант	Інтеграційне	Реалізовано механізм кешування локалізації через Cache::remember, що дозволило знизити кількість SQL-запитів на головній сторінці на 40%.

Продовження таблиці 3.1

Некоректне відображення іконок Lucide у динамічно завантажених елементах (модалки, акордеони)	UI Testing	У функцію handleDoctorModal та інші модулі додано примусовий виклик initIcons() після кожної маніпуляції з DOM-структурою.
Низька швидкість фільтрації списку лікарів при великій кількості спеціалізацій	Навантажувальне	Оптимізовано логіку initBookingFilter у app.ts, впроваджено пряму маніпуляцію опціями select замість повного перерендеру елементів, що пришвидшило відгук інтерфейсу.
Втрата обраної теми оформлення після повного очищення сесії користувача	Usability Testing	Логіку збереження теми перенесено з сесії сервера у localStorage браузера, що забезпечило консистентність інтерфейсу незалежно від стану серверної авторизації.

Заключна фаза тестування включала аналіз безпеки та цілісності даних. Перевірка механізму localStorage для збереження обраної теми показала, що дані не втрачаються після закриття вкладки або перезавантаження браузера. Також було проведено тестування форм на предмет стійкості до XSS-атак: спроби введення скриптів у поля імені пацієнта виявилися марними, оскільки Blade-шаблонізатор Laravel автоматично екранує всі вихідні дані, а TypeScript-логіка модальних вікон використовує властивість textContent замість innerHTML для динамічних даних. Загальний аналіз результатів тестування дозволяє стверджувати, що розроблена система є стабільною, відповідає всім поставленим функціональним вимогам та готова до експлуатації в реальних умовах медичного закладу. Усі виявлені під час розробки дрібні дефекти верстки та логіки були оперативно усунуті, що підтверджується фінальним циклом регресійного тестування.

ВИСНОВКИ

У даній кваліфікаційній роботі було розроблено програмну систему “Lanka”, призначену для відстеження гуманітарних вантажів у логістичних ланцюгах постачання з використанням технології блокчейн. Головною метою роботи було створення інструменту, що забезпечує підвищення прозорості, підзвітності та ефективності процесів доставки гуманітарної допомоги шляхом інтеграції децентралізованих рішень з традиційними веб-технологіями.

У ході виконання роботи було вирішено наступні ключові завдання:

- проведено глибокий аналіз проблем у сфері логістики гуманітарної допомоги, зокрема виявлено системну відсутність прозорості, високі корупційні ризики, неефективне управління ресурсами та складність координації дій учасників;
- досліджено існуючі технологічні рішення для відстеження поставок, з особливим акцентом на перевагах та викликах застосування блокчейн-технологій, що дозволило обґрунтувати вибір платформи Ethereum та децентралізованого сховища IPFS;
- сформульовано детальні функціональні та нефункціональні вимоги до системи “Lanka”, що стали основою для подальшого проєктування та розробки;
- розроблено гібридну архітектуру системи, яка оптимально поєднує переваги централізованої бази даних PostgreSQL для зберігання оперативної інформації та блокчейну Ethereum для фіксації незмінних аудиторських даних та ключових подій;
- спроектовано модульну структуру, що включає блокчейн-компоненти, бекенд-логіку та фронтенд-інтерфейс;
- спроектовано структуру даних для реляційної бази даних та розроблено набір смарт-контрактів на мові Solidity (UserRegistry, AidPackage, DocumentRegistry, AuditTrail), що реалізують основну децентралізовану логіку системи;

– реалізовано ключові модулі системи “Lanka”: блокчейн-компоненти розгорнуто у тестовій мережі Ethereum (Sepolia); розроблено бекенд з використанням Next.js API Routes, Prisma ORM для взаємодії з PostgreSQL та Ethers.js для інтеграції з блокчейном; створено інтуїтивно зрозумілий користувацький фронтенд-інтерфейс на Next.js/React/TypeScript з інтеграцією RainbowKit для взаємодії з Ethereum-гаманцями.

Також важливо зазначити, що проведено комплексне тестування системи, що включало модульне, інтеграційне, системне тестування та тестування зручності використання. За результатами тестування було виявлено та усунено 26 дефектів різного рівня пріоритетності, що підтвердило функціональну повноту, стабільність та надійність розробленого рішення.

Наукова новизна роботи полягає у розробці специфічної гібридної архітектури для системи відстеження гуманітарних вантажів, що ефективно інтегрує централізовані бази даних для оперативності та блокчейн-технології (Ethereum, IPFS) для забезпечення незмінності, прозорості та аудиту ключових даних. Також елементом новизни є створення унікального набору смарт-контрактів, оптимізованих для потреб гуманітарної логістики, що дозволяють децентралізовано керувати ролями учасників, життєвим циклом вантажів та реєстром супровідних документів.

Практичне значення отриманих результатів полягає у створенні функціонального прототипу програмної системи “Lanka”. Розроблена система демонструє можливість суттєвого підвищення рівня прозорості в ланцюгах постачання гуманітарної допомоги, мінімізації ризиків шахрайства та нецільового використання ресурсів. Інтеграція з блокчейном забезпечує створення незмінного аудиторського сліду, доступного для всіх авторизованих учасників, що сприяє зміцненню довіри між донорами, логістичними операторами та отримувачами допомоги.

Система також може бути взята за основу для подальшого розвитку та впровадження гуманітарними організаціями, волонтерськими ініціативами та державними установами для покращення процесів доставки, розподілу та

контролю гуманітарної допомоги, що є особливо актуальним в умовах кризових ситуацій та міжнародних гуманітарних місій.

Подальшими напрямками розвитку проекту можуть бути:

- Інтеграція з L2-рішеннями для Ethereum з метою зниження вартості транзакцій та підвищення їх швидкості.
- Розширення функціоналу для аналітики та звітності на основі зібраних даних.
- Інтеграція з IoT-пристроями для автоматичного збору даних про місцезнаходження та стан вантажів.
- Проведення пілотного впровадження системи в реальних умовах для отримання зворотного зв'язку та подальшого вдосконалення.

Таким чином, поставлена мета кваліфікаційної роботи – розробка програмної системи відстеження гуманітарних вантажів у логістичних ланцюгах постачання на основі блокчейн-технологій – була досягнута. Розроблена система “Lanka” є сучасним та перспективним рішенням, що має потенціал для значного покращення ефективності та прозорості гуманітарних операцій.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Що таке блокчейн: технологія, яка змінить все [Електронний ресурс] – Режим доступу: https://dou.ua/lenta/articles/what-is-blockchain/ (дата звернення – 15.01.2025). – Назва з екрану.
2. What is Ethereum? [Електронний ресурс] – Режим доступу: https://ethereum.org/en/what-is-ethereum/ (дата звернення – 18.01.2025). – Назва з екрану.
3. Solidity Language Documentation [Електронний ресурс] – Режим доступу: https://docs.soliditylang.org/en/v0.8.20/ (дата звернення – 20.01.2025). – Назва з екрану.
4. Introduction to Smart Contracts [Електронний ресурс] – Режим доступу: https://ethereum.org/en/developers/docs/smart-contracts/ (дата звернення – 22.01.2025). – Назва з екрану.
5. IPFS Docs [Електронний ресурс] – Режим доступу: https://docs.ipfs.tech/ (дата звернення – 25.01.2025). – Назва з екрану.
6. Гуманітарна логістика: що це і чому так важливо для України [Електронний ресурс] – Режим доступу: https://www.epravda.com.ua/columns/2022/04/12/685625/ (дата звернення – 28.01.2025). – Назва з екрану.
7. Next.js by Vercel - The React Framework [Електронний ресурс] – Режим доступу: https://nextjs.org/docs (дата звернення – 02.02.2025). – Назва з екрану.
8. React Official Documentation [Електронний ресурс] – Режим доступу: https://react.dev/ (дата звернення – 05.02.2025). – Назва з екрану.
9. TypeScript Official Documentation [Електронний ресурс] – Режим доступу: https://www.typescriptlang.org/docs/ (дата звернення – 08.02.2025). – Назва з екрану.
10. Tailwind CSS Documentation [Електронний ресурс] – Режим доступу: https://tailwindcss.com/docs/installation (дата звернення – 10.02.2025). – Назва з екрану.

11. Prisma Docs [Електронний ресурс] – Режим доступу: https://www.prisma.io/docs/ (дата звернення – 12.02.2025). – Назва з екрану.

12. PostgreSQL Documentation [Електронний ресурс] – Режим доступу: https://www.postgresql.org/docs/current/index.html (дата звернення – 15.02.2025). – Назва з екрану.

13. Hardhat Documentation [Електронний ресурс] – Режим доступу: https://hardhat.org/docs (дата звернення – 18.02.2025). – Назва з екрану.

14. Ethers.js Documentation [Електронний ресурс] – Режим доступу: https://docs.ethers.org/v5/ (дата звернення – 20.02.2025). – Назва з екрану.

15. Docker Documentation [Електронний ресурс] – Режим доступу: https://docs.docker.com/ (дата звернення – 23.02.2025). – Назва з екрану.

16. Git Documentation [Електронний ресурс] – Режим доступу: https://git-scm.com/doc (дата звернення – 26.02.2025). – Назва з екрану.

17. GitHub Actions Documentation [Електронний ресурс] – Режим доступу: https://docs.github.com/en/actions (дата звернення – 01.03.2025). – Назва з екрану.

18. Jest Official Documentation [Електронний ресурс] – Режим доступу: https://jestjs.io/docs/getting-started (дата звернення – 04.03.2025). – Назва з екрану.

19. React Testing Library Docs [Електронний ресурс] – Режим доступу: https://testing-library.com/docs/react-testing-library/intro/ (дата звернення – 07.03.2025). – Назва з екрану.

20. Playwright Documentation [Електронний ресурс] – Режим доступу: https://playwright.dev/docs/intro (дата звернення – 10.03.2025). – Назва з екрану.

21. 10 Usability Heuristics for User Interface Design [Електронний ресурс] – Режим доступу: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення – 13.03.2025). – Назва з екрану.

22. Manifesto for Agile Software Development [Електронний ресурс] – Режим доступу: https://agilemanifesto.org/ (дата звернення – 16.03.2025). – Назва з екрану.

23. Що таке криптографія? [Електронний ресурс] – Режим доступу: https://academy.binance.com/uk/articles/what-is-cryptography (дата звернення – 19.03.2025). – Назва з екрану.

24. IoT in Logistics: Benefits, Use Cases & Challenges [Електронний ресурс] – Режим доступу: https://www.scnsoft.com/blog/iot-in-logistics (дата звернення – 22.03.2025). – Назва з екрану.

25. Radix Primitives Documentation [Електронний ресурс] – Режим доступу: https://www.radix-ui.com/primitives/docs/overview/introduction (дата звернення – 25.03.2025). – Назва з екрану.

26. Recharts API Documentation [Електронний ресурс] – Режим доступу: https://recharts.org/en-US/api (дата звернення – 28.03.2025). – Назва з екрану.

27. TanStack Table v8 Docs [Електронний ресурс] – Режим доступу: https://tanstack.com/table/v8/docs/introduction (дата звернення – 01.04.2025). – Назва з екрану.

28. SWR Documentation [Електронний ресурс] – Режим доступу: https://swr.vercel.app/docs/getting-started (дата звернення – 04.04.2025). – Назва з екрану.

29. TanStack Query v5 Docs [Електронний ресурс] – Режим доступу: https://tanstack.com/query/v5/docs/react/overview (дата звернення – 07.04.2025). – Назва з екрану.

30. Blockchain in Supply Chain Management: A Review, Opportunities, and Challenges [Електронний ресурс] – Режим доступу: https://www.mdpi.com/2071-1050/11/18/4868 (дата звернення – 10.04.2025). – Назва з екрану.

31. OpenZeppelin Contracts Documentation [Електронний ресурс] – Режим доступу: https://docs.openzeppelin.com/contracts/4.x/ (дата звернення – 13.04.2025). – Назва з екрану.

32. What are DApps? A Guide to Decentralized Applications [Електронний ресурс] – Режим доступу: https://consensys.io/blog/developers/what-are-dapps-a-guide-to-decentralized-applications (дата звернення – 16.04.2025). – Назва з екрану.

33. RainbowKit Documentation [Електронний ресурс] – Режим доступу: https://www.rainbowkit.com/docs/introduction (дата звернення – 19.04.2025). – Назва з екрану.

34. Системний аналіз: основи теорії та практики [Електронний ресурс] – Режим доступу: https://ela.kpi.ua/bitstream/123456789/20134/1/System_analiz_Navch_posib_2017.pdf (дата звернення – 22.04.2025). – Назва з екрану.

35. Transparency in Humanitarian Logistics: A Systematic Literature Review [Електронний ресурс] – Режим доступу: https://www.researchgate.net/publication/338923736_Transparency_in_Humanitarian_Logistics_A_Systematic_Literature_Review (дата звернення – 25.04.2025). – Назва з екрану.

36. Can Blockchain Technology Revolutionize Humanitarian Aid Delivery? [Електронний ресурс] – Режим доступу: https://www.weforum.org/agenda/2018/01/blockchain-technology-revolutionize-humanitarian-aid-delivery/ (дата звернення – 28.04.2025). – Назва з екрану.

37. BPMN 2.0 Tutorial [Електронний ресурс] – Режим доступу: https://camunda.com/bpmn/tutorial/ (дата звернення – 01.05.2025). – Назва з екрану.

38. NFTs in Supply Chain: Use Cases & Benefits [Електронний ресурс] – Режим доступу: https://101blockchains.com/nft-in-supply-chain/ (дата звернення – 04.05.2025). – Назва з екрану.

ДОДАТОК А

(обов’язковий)

ЛІСТИНГИ ПРОГРАМНОГО КОДУ

```
Vladyslav Vasiliev
class BookingController extends Controller
{
    Vladyslav Vasiliev
    public function index()
    {

        $specializations = Specialization::withTranslations()->visible()->positionSorted()->get();
        $doctors = Doctor::query()->get();
        $services = Service::withTranslations()->visible()->positionSorted()->get();

        return view( view: 'pages.booking', compact([
            'specializations',
            'doctors',
            'services',
        ]));
    }

    Vladyslav Vasiliev
    public function store(StoreAppointmentRequest $request)
    {
        $data = $request->validated();

        if (auth()->id()) {
            $data['user_id'] = auth()->id();
        }

        $data['status'] = 'pending';

        $appointment = Appointment::create($data);

        return redirect()->route( route: 'auth.profile')
            ->with('success', 'Ви успішно записані на прийом! Ми зателефонуємо для підтвердження.');
    }
}
```

Рисунок А.1. – Веб-контролер по обробці записів

```php
public function withValidator($validator)
{
    $validator->after(function ($validator) {
        $doctorId = $this->input( key: 'doctor_id');
        $serviceId = $this->input( key: 'service_id');
        $requestedStart = Carbon::parse($this->input( key: 'appointment_at'));

        if (!$doctorId || !$serviceId || !$requestedStart) return;

        $service = Service::find($serviceId);
        $duration = $service->duration ?? 30; // за замовчуванням 30 хв
        $requestedEnd = $requestedStart->copy()->addMinutes($duration);

        $dayOfWeek = $requestedStart->dayOfWeekIso; // 1 (Пн) - 7 (Нд)
        $schedule = Schedule::where('doctor_id', $doctorId)
            ->where('day_of_week', $dayOfWeek)
            ->first();

        if (!$schedule) {
            $validator->errors()->add('appointment_at', "Лікар у цей день не працює");
            return;
        }
        if ($requestedStart->format( format: 'H:i:s') < $schedule->start_time ||
            $requestedEnd->format( format: 'H:i:s') > $schedule->end_time) {

            $validator->errors()->add('appointment_at', "Лікар у цей час не працює. Графік на цей день: {$schedule->start_time} - {$schedule->end_time}");
            return;
        }

        $conflict = Appointment::where('doctor_id', $doctorId)
            ->whereIn('status', [Appointment::STATUS_PENDING, Appointment::STATUS_CONFIRMED])
            ->where(function ($query) use ($requestedStart, $requestedEnd) {
                $query->where(function ($q) use ($requestedStart, $requestedEnd) {
                    $q->where('appointment_at', '>=', $requestedStart)
                        ->where('appointment_at', '<', $requestedEnd);
                })->orWhere(function ($q) use ($requestedStart, $requestedEnd) {
                    $q->whereHas('service', function($sub) use ($requestedStart) {
                    });
                });
            })
            ->first();

        $dayAppointments = Appointment::where('doctor_id', $doctorId)
            ->whereDate('appointment_at', $requestedStart->toDateString())
            ->whereIn('status', ['pending', 'confirmed'])
            ->with('service')

        foreach ($dayAppointments as $existing) {
            $existStart = Carbon::parse($existing->appointment_at);
            $existEnd = $existStart->copy()->addMinutes( value: $existing->service->duration ?? 30);

            if ($requestedStart->lt($existEnd) && $requestedEnd->gt($existStart)) {

                $suggestedTime = $existEnd->format( format: 'H:i');

                $validator->errors()->add('appointment_at',
                    "Цей час зайнятий (запис до {$existEnd->format( format: 'H:i')}). Спробуйте записатися на {$suggestedTime} або інший час."
                );
                return;
            }
        }
    });
    }
}
```

Рисунок А.2. – Валідація на дату і час запису до лікаря

```php
class AdminMiddleware
{
    /**
     * Handle an incoming request.
     *
     * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)  $next
     */
    Vladyslav Vasiliev
    public function handle(Request $request, Closure $next): Response
    {
        if (!auth()->check()) {
            return redirect(route( name: 'admin.login'));
        }

        $user = auth()->user();

        if (!app()->runningInConsole() && $user) {
            $roles = Role::with( relations: 'permissions')->get();

            foreach ($roles as $role) {
                foreach ($role->permissions as $permissions) {
                    $permissionsArray[$permissions->slug][] = $role->id;
                }
            }

            if (empty($permissionsArray)) {
                Auth::logout();
                $request->session()->invalidate();
                $request->session()->regenerateToken();
                return redirect(route( name: 'admin.login'));
            } else {
                foreach ($permissionsArray as $slug => $roles) {
                    Gate::define($slug, function (User $user) use ($roles) {
                        return count(array_intersect($user->roles->pluck('id')->toArray(), $roles)) > 0;
                    });
                }
            }
        }
        return $next($request);
    }
}
```

Рисунок А.3. – Перевірка запитів на права адміністратора

ДОДАТОК Б

(обов’язковий)

ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

Б.1 Код шляхів сайту та до панелі адміністратора

```
<?php

use App\Http\Controllers\Web\BookingController;
use App\Http\Controllers\Web\ContactsController;
use App\Http\Controllers\Web\DoctorController;
use App\Http\Controllers\Web\HomePageController;
use App\Http\Controllers\Web\LocaleController;
use App\Http\Controllers\Web\SpecializationController;
use App\Http\Middleware\WebLocaleMiddleware;
use Illuminate\Support\Facades\Route;

Route::get('locale/{locale}', LocaleController::class)->name('locale');

Route::group([
    'middleware' => WebLocaleMiddleware::class,
    'prefix' => WebLocaleMiddleware::getLocale(),
], function () {
    Route::get('/', HomePageController::class)->name('home');
    Route::get('booking', [BookingController::class, 'index'])->name('booking');
    Route::post('booking', [BookingController::class, 'store'])->name('appointments.store');
    Route::get('contacts', ContactsController::class)->name('contacts');
    Route::get('doctors', [DoctorController::class, 'index'])->name('doctors');
    Route::get('specializations', [SpecializationController::class, 'index'])->name('specializations');
    Route::get('specializations/{specialization:slug}', [SpecializationController::class, 'show'])-
>name('services');
    Route::view('about', 'pages.about')->name('about');
    Route::view('login', 'pages.login')->name('login');
    Route::view('register', 'pages.register')->name('register');
}
);
```

```
<?php

use App\Http\Controllers\Admin\AppointmentController;
use App\Http\Controllers\Admin\AuthController;
use App\Http\Controllers\Admin\DoctorController;
use App\Http\Controllers\Admin\HomeController;
use App\Http\Controllers\Admin\PermissionsController;
use App\Http\Controllers\Admin\RoleController;
use App\Http\Controllers\Admin\ScheduleController;
use App\Http\Controllers\Admin\ServiceController;
```

```
use App\Http\Controllers\Admin\SpecializationController;
use App\Http\Controllers\Admin\TranslationController;
use App\Http\Controllers\Admin\UserController;
use App\Http\Controllers\Admin\VariableController;
use Illuminate\Support\Facades\Route;

Route::get('/', [HomeController::class, 'index'])->name('home');

Route::group(['middleware' => 'only-ajax'], function () {
    Route::post('specializations/{id}/ajax_field', [SpecializationController::class, 'ajaxFieldChange'])->name('specializations.ajax_field');
    Route::post('services/{id}/ajax_field', [ServiceController::class, 'ajaxFieldChange'])->name('services.ajax_field');
});

Route::resource('specializations', SpecializationController::class);
Route::resource('doctors', DoctorController::class);
Route::resource('services', ServiceController::class);
Route::resource('schedules', ScheduleController::class);
Route::resource('appointments', AppointmentController::class);

Route::resource('permissions', PermissionsController::class);
Route::resource('roles', RoleController::class);
Route::resource('users/{type}', UserController::class)->parameter('{type}', 'user')->names('users');

Route::get('login', [AuthController::class, 'loginForm'])->name('login')->withoutMiddleware('admin');
Route::post('login', [AuthController::class, 'login'])->name('login')->withoutMiddleware('admin');
Route::post('logout', [AuthController::class, 'logout'])->name('logout');

Route::get('locale/{locale}', [HomeController::class, 'locale'])->name('locale');

// variables
Route::get('variables/list', [VariableController::class, 'list'])
    ->name('variables.list.index');
Route::post('variables/{variable}/status', [VariableController::class, 'updateStatus']);
Route::resource('variables', VariableController::class)
    ->except('show');

// translations
Route::get('translation/{group}', [TranslationController::class, 'index'])->name('translation.index');

Route::post('translation/{group}', [TranslationController::class, 'update'])
    ->name('translation.update');

Route::post('translations/upload', [TranslationController::class, 'uploadFromFile'])
    ->name('translations.upload');

Route::get('translations/conflict', [TranslationController::class, 'conflicts'])
    ->name('translations.conflicts');
```

```
Route::put('translations/conflict/resolve', [TranslationController::class, 'resolveConflicts'])
   ->name('translations.conflicts.resolve');

Route::post('translations/conflict/force', [TranslationController::class, 'forceMergeConflicts'])
   ->name('translations.conflicts.force');

Route::post('translations/delete', [TranslationController::class, 'destroy'])-
>name('translations.delete');
```

Б.2 Код форми логіна для Адміністрування

```
<?php

namespace App\Http\Controllers\Admin;

use App\Http\Controllers\Controller;
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Redirect;
use Illuminate\Support\MessageBag;

class AuthController extends Controller
{
   public function __construct()
   {
      $this->middleware('guest')->except('logout');
   }

   public function loginForm()
   {
      return view('adminlte::auth.login');
   }

   public function login(Request $request)
   {
      $email = $request->email;

      $user = User::where('email', $email)
         ->first();

      if (!$user) {
         $errors = new MessageBag(['email' => ['These credentials do not match our records.']]);

         return Redirect::back()->withErrors($errors);
      }

      $is_valid = Auth::attempt(
```

```
            [
                'email' => $request->email,
                'password' => $request->password
            ],
            $request->boolean('remember')
        );

        if ($is_valid) {
            $request->session()->regenerate();

            return redirect()->intended(route('admin.home'));
        }

        $errors = new MessageBag(['email' => ['These credentials do not match our records.']]);

        return Redirect::back()->withErrors($errors);
    }

    public function logout(Request $request)
    {
        Auth::logout();

        $request->session()->invalidate();

        $request->session()->regenerateToken();

        return redirect(route('admin.login'));
    }
}
```

Б.3 Код контролера адміністративної панелі по обробці спеціальностей

```
<?php

namespace App\Models;

use App\Models\Traits\PositionSortedTrait;
use App\Models\Traits\VisibleTrait;
use App\Models\Traits\WithTranslationsTrait;
use Astrotomic\Translatable\Translatable;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Specialization extends Model
{
    use HasFactory;
    use Translatable;
    use WithTranslationsTrait;
    use PositionSortedTrait;
    use VisibleTrait;
```

```
    protected $translatedAttributes = [
        'title',
    ];

    protected $fillable = [
        'slug',
        'image',
        'icon',
        'status',
        'position',
    ];

    public function doctors(): \Illuminate\Database\Eloquent\Relations\HasMany
    {
        return $this->hasMany(Doctor::class);
    }

    public function services(): \Illuminate\Database\Eloquent\Relations\HasMany
    {
        return $this->hasMany(Service::class);
    }
}

<?php

declare(strict_types=1);

namespace App\Http\Controllers\Admin;

use App\DataTables\SpecializationDataTable;
use App\Http\Controllers\Traits\AjaxFieldsChangerTrait;
use App\Http\Controllers\Traits\FileProcessorTrait;
use App\Http\Requests\Admin\Specialization\SpecializationCreateRequest;
use App\Http\Requests\Admin\Specialization\SpecializationUpdateRequest;
use App\Models\Specialization;
use Gate;

class SpecializationController extends AdminBaseController
{
    use AjaxFieldsChangerTrait;
    use FileProcessorTrait;

    public string $module = 'specializations';

    protected function getModelClass(): string
    {
        return Specialization::class;
    }

    public function index(SpecializationDataTable $dataTable)
```

```
    {
        abort_unless(Gate::allows($this->module . '_access'), 403);

        $data['module'] = $this->module;

        return $this->renderDatatable($dataTable, $data);
    }

    public function store(SpecializationCreateRequest $request)
    {
        abort_unless(Gate::allows($this->module . '_create'), 403);

        $data = $request->validated();

        $data = $this->processFile(
            model: null,
            field:'image',
            remove_field: 'isRemoveImage',
            module: 'specializations',
            folder: 'images',
            data: $data,
        );

        $data = $this->processFile(
            model: null,
            field:'icon',
            remove_field: 'isRemoveIcon',
            module: 'specializations',
            folder: 'icons',
            data: $data,
        );

        Specialization::create($data);

        flash()->success(message: __('admin_labels.success.add', ['model' => __('admin_labels.' .
\Str::singular($this->module))]), title: __('admin_labels.success_label'));

        return redirect()->route('admin.' . $this->module . '.index');
    }

    public function update(SpecializationUpdateRequest $request, Specialization $specialization)
    {
        abort_unless(Gate::allows($this->module . '_edit'), 403);

        $data = $request->validated();

        $data = $this->processFile(
            model: $specialization,
            field:'image',
            remove_field: 'isRemoveImage',
            module: 'specializations',
```

```
            folder: 'images',
            data: $data,
        );

        $data = $this->processFile(
            model: $specialization,
            field:'icon',
            remove_field: 'isRemoveIcon',
            module: 'specializations',
            folder: 'icons',
            data: $data,
        );

        $specialization->update($data);

        flash()->success(message: __('admin_labels.success.update', ['model' => __('admin_labels.' .
\Str::singular($this->module))]), title: __('admin_labels.success_label'));

        return redirect()->route('admin.' . $this->module . '.index');
    }
}

<!DOCTYPE html>
<html lang="uk" data-theme="light">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>MedCenter | Професійна Турбота</title>
    <script src="https://unpkg.com/lucide@latest"></script>
    @vite([ 'resources/css/web/app.scss' ])
    @stack('styles')
</head>
<body>

<x-layout.header/>

<main>
    {{ $slot  }}
</main>

<x-layout.footer/>

@stack('scripts')
@vite([ 'resources/js/web/app.ts' ])
</body>
</html>

@include('admin.partials._buttons', ['class' => 'buttons-top'])
<br>
<div class="card">
    <div class="card-header">
```

```
      @if($model->id)
         {{__('admin_labels.edit')}}
      @else
         {{__('admin_labels.add')}}
      @endif
   </div>
   <div class="row">
      <div class="col-md-12">
         <div class="card-body">
            <ul class="nav nav-tabs " id="custom-content-below-tab" role="tablist">
               @include('admin.partials.translate_nav_tabs')

               <li class="nav-item">
                  <a class="nav-link" data-toggle="pill" href="#general" role="tab" aria-
controls="general"
                     aria-selected="false">{{__('admin_labels.tab_general')}}</a>
               </li>
            </ul>
            <div class="tab-content pt-4 pb-4 pl-2 pr-2">
               @foreach (config('app.locales') as $key => $locale)
                  <div class="tab-pane fade @if($key == 0)active show @endif" id="tab_{!! $locale
!!}"
                     role="tabpanel"
                     aria-labelledby="tab_{!! $locale !!}">
                     @include('admin.views.' . $module . '.tabs.locale', array('errors' => $errors , 'locale'
=> $locale))
                  </div>
               @endforeach

               <div class="tab-pane" id="general">
                  @include('admin.views.' . $module . '.tabs.general')
               </div>
            </div>

         </div>
      </div>
   </div>
</div>
@include('admin.partials._buttons')

@include('admin.fields.slug', ['model' => $model])

@include(
   'admin.fields.default_image',
   [
      'title' => 'image',
      'model' => $model,
      'field' => 'image',
      'remove_field' => 'isRemoveImage',
   ]
)
```

```
@include(
    'admin.fields.default_image',
    [
        'title' => 'icon',
        'model' => $model,
        'field' => 'icon',
        'remove_field' => 'isRemoveIcon',
    ]
)

@include(
    'admin.fields.select',
    [
        'title' => 'status',
        'model' => $model,
        'field' => 'status',
        'values' => [0 => __('admin_labels.no'), 1 => __('admin_labels.yes')],
    ]
)

@include(
    'admin.fields.numeric_input',
    [
        'title' => 'position',
        'model' => $model,
        'field' => 'position',
    ]
)
```

Б.4 Код для обробки клієнтської частини

```
<?php

namespace App\Http\Controllers\Web;

use App\Http\Controllers\Controller;
use App\Models\Doctor;
use App\Models\Specialization;

class HomePageController extends Controller
{
    public function __invoke()
    {
        $specializations = Specialization::withTranslations()->visible()->positionSorted()->get();
        $doctors = Doctor::query()->limit(3)->get();

        return view('pages.home', compact([
            'specializations',
            'doctors',
```

```
        ]));
    }
}

<x-layout>
    <section class="hero">
        <div class="container hero-grid">
            <div class="hero-content">
                <h1>{{ __('site_labels.your_health_our_main_priority') }}</h1>
                <p>{{ __('site_labels.modern_clinic') }}</p>
                <div class="hero-btns">
                    <a href="#specs" class="btn-primary">{{ __('site_labels.our_services') }}</a>
                    <a href="#about" class="btn-outline">{{ __('site_labels.more') }}</a>
                </div>
            </div>
            <div class="hero-image">
                <div class="image-placeholder">
                    <img class="image-placeholder" src="{{ asset('images/home.jpg') }}" alt="Hero
Image">
                </div>
            </div>
        </div>
    </section>

    <section id="about" class="stats-section">
        <div class="container">
            <div class="stats-grid">
                <div class="stat-item">
                    <h3>10+</h3>
                    <p>{{ __('site_labels.years_of_experience') }}</p>
                </div>
                <div class="stat-item">
                    <h3>25+</h3>
                    <p>{{ __('site_labels.professional_doctors') }}</p>
                </div>
                <div class="stat-item">
                    <h3>15к+</h3>
                    <p>{{ __('site_labels.satisfied_patients') }}</p>
                </div>
            </div>
        </div>
    </section>

    <section id="specs" class="carousel-section">
        <div class="container">
            <h2 class="section-title">{{ __('site_labels.our_specializations') }}</h2>
            <div class="carousel-container">
                <button class="carousel-btn prev"><i data-lucide="chevron-left"></i></button>
                <div class="carousel-track">
                    @foreach ($specializations as $specialization)
                        <div class="spec-card">
```

```
                    <img src="{{ file_url($specialization->icon) }}" alt="{{$specialization->title}}"
class="spec-icon-img">
                    <h4>{{$specialization->title}}</h4>
                </div>
            @endforeach
        </div>
        <button class="carousel-btn next"><i data-lucide="chevron-right"></i></button>
      </div>
    </div>
  </section>

  <section id="doctors" class="doctors-section">
    <div class="container">
      <h2 class="section-title">{{ __('site_labels.our_doctors') }}</h2>
      <div class="doctors-grid">
        @foreach ($doctors as $doctor)
          <div class="doctor-card">
            <div class="doc-img">
              <img class="doc-img" src="{{ file_url($doctor->image) }}" alt="{{ $doctor-
>name }}">
            </div>
            <div class="doc-info">
              <h4>{{ $doctor->name }}</h4>
              <span>{!! $doctor->description !!}</span>
              @php
                $string = strip_tags($doctor->biography);
                if (strlen($string) > 200) {

                  $stringCut = substr($string, 0, 200);
                  $endPoint = strrpos($stringCut, ' ');

                  $string = $endPoint? substr($stringCut, 0, $endPoint) : substr($stringCut, 0);
                }
              @endphp
              <p>{!! $string !!}</p>
            </div>
          </div>
        @endforeach
      </div>
    </div>
  </section>
</x-layout>

interface LucideStatic {
  createIcons: () => void;
}
declare const lucide: LucideStatic;

interface DoctorSchedule {
  day_of_week: number;
  start_time: string;
```

```
    end_time: string;
}

const weekDays = ['Пн', 'Вт', 'Ср', 'Чт', 'Пт', 'Сб', 'Нд'];

/**
 * Ініціалізація іконок
 */
const initIcons = (): void => {
    if (typeof lucide !== 'undefined') lucide.createIcons();
};

initIcons();

/**
 * Тема та Navbar
 */
const htmlElement = document.documentElement;
const themeToggle = document.getElementById('theme-toggle') as HTMLButtonElement | null;
const themeIcon = document.getElementById('theme-icon');

const toggleTheme = (): void => {
    if (!themeToggle || !themeIcon) return;
    const currentTheme = htmlElement.getAttribute('data-theme');
    const newTheme = currentTheme === 'light' ? 'dark' : 'light';
    htmlElement.setAttribute('data-theme', newTheme);
    localStorage.setItem('theme', newTheme);
    themeIcon.setAttribute('data-lucide', newTheme === 'dark' ? 'sun' : 'moon');
    initIcons();
};

themeToggle?.addEventListener('click', toggleTheme);

window.addEventListener('scroll', () => {
    const navbar = document.querySelector('.navbar') as HTMLElement | null;
    if (!navbar) return;
    if (window.scrollY > 50) {
        navbar.style.boxShadow = '0 4px 20px rgba(0,0,0,0.1)';
        navbar.style.padding = '10px 0';
    } else {
        navbar.style.boxShadow = 'none';
        navbar.style.padding = '15px 0';
    }
});

/**
 * Карусель
 */
const track = document.querySelector('.carousel-track') as HTMLElement | null;
const nextBtn = document.querySelector('.next');
const prevBtn = document.querySelector('.prev');
```

```
const scrollCarousel = (direction: 'left' | 'right'): void => {
    if (!track) return;
    const scrollAmount = 300;
    track.scrollBy({ left: direction === 'right' ? scrollAmount : -scrollAmount, behavior: 'smooth' });
};

nextBtn?.addEventListener('click', () => scrollCarousel('right'));
prevBtn?.addEventListener('click', () => scrollCarousel('left'));

/**
 * МОДАЛКА ЛІКАРІВ (ВИПРАВЛЕНА ЛОГІКА)
 */
const modal = document.getElementById('doctor-modal');
const scheduleList = document.getElementById('modal-schedule-list');

const handleDoctorModal = () => {
    const openBtns = document.querySelectorAll('.open-doc-modal, .open-doctor-modal'); // Підтримка обох класів
    const closeBtn = document.querySelector('.close-modal');

    openBtns.forEach(button => {
        button.addEventListener('click', () => {
            const btn = button as HTMLElement;
            const modalName = document.getElementById('modal-name');
            const modalSpec = document.getElementById('modal-spec');
            const modalStory = document.getElementById('modal-story');

            if (modalName) modalName.textContent = btn.dataset.name || btn.getAttribute('data-name') || '';
            if (modalSpec) modalSpec.textContent = btn.dataset.spec || btn.getAttribute('data-spec') || '';
            if (modalStory) modalStory.textContent = btn.dataset.story || btn.getAttribute('data-story') || '';

            // Обробка графіка
            if (scheduleList) {
                scheduleList.innerHTML = '';
                try {
                    const rawSchedule = btn.dataset.schedule || btn.getAttribute('data-schedule');
                    const schedules: DoctorSchedule[] = JSON.parse(rawSchedule || '[]');
                    const scheduleMap = new Map();

                    schedules.forEach(s => {
                        const start = s.start_time.substring(0, 5);
                        const end = s.end_time.substring(0, 5);
                        scheduleMap.set(Number(s.day_of_week), `${start} - ${end}`);
                    });

                    for (let i = 1; i <= 7; i++) {
                        const time = scheduleMap.get(i);
                        const isOff = !time;
                        const item = document.createElement('div');
```

```
                item.className = `schedule-item ${isOff ? 'off' : ''}`;
                item.innerHTML = `
                    <span class="day-name">${weekDays[i - 1]}</span>
                    <span class="time-range">${isOff ? 'Вихідний' : time}</span>
                `;
                scheduleList.appendChild(item);
            }
        } catch (e) {
            console.error("Schedule error:", e);
            scheduleList.innerHTML = '<p>Графік недоступний</p>';
        }
    }

    modal?.classList.add('active');
    document.body.style.overflow = 'hidden';
    initIcons(); // Перемальовуємо X у модалці
  });
});

const closeActions = () => {
  modal?.classList.remove('active');
  document.body.style.overflow = '';
};

closeBtn?.addEventListener('click', closeActions);
modal?.addEventListener('click', (e) => { if (e.target === modal) closeActions(); });
};

/**
 * Фільтрація запису (Booking)
 */
const initBookingFilter = () => {
  const specFilter = document.getElementById('spec_filter') as HTMLSelectElement;
  const doctorSelect = document.getElementById('doctor_id') as HTMLSelectElement;
  const serviceSelect = document.getElementById('service_id') as HTMLSelectElement;

  if (!specFilter || !doctorSelect || !serviceSelect) return;

  const allDoctorOptions = Array.from(doctorSelect.options);
  const allServiceOptions = Array.from(serviceSelect.options);

  specFilter.addEventListener('change', () => {
    const selectedSpecId = specFilter.value;
    const update = (select: HTMLSelectElement, options: HTMLOptionElement[]) => {
      const val = select.value;
      select.innerHTML = '';
      options.filter(opt => {
        const sid = opt.getAttribute('data-spec');
        return selectedSpecId === 'all' || !sid || sid === selectedSpecId;
      }).forEach(opt => select.add(opt));
      if (![...select.options].some(o => o.value === val)) select.selectedIndex = 0;
```

```
        };
        update(doctorSelect, allDoctorOptions);
        update(serviceSelect, allServiceOptions);
    });
};

/**
 * Burger Menu
 */
const initBurger = () => {
    const menuToggle = document.querySelector('.menu-toggle');
    const navMenu = document.querySelector('.nav-menu');
    if (!menuToggle || !navMenu) return;

    menuToggle.addEventListener('click', () => {
        navMenu.classList.toggle('active');
        menuToggle.classList.toggle('is-active');
        document.body.style.overflow = navMenu.classList.contains('active') ? 'hidden' : '';
    });

    document.querySelectorAll('.nav-menu a').forEach(link => {
        link.addEventListener('click', () => {
            navMenu.classList.remove('active');
            menuToggle.classList.remove('is-active');
            document.body.style.overflow = '';
        });
    });
};

/**
 * Запуск усього
 */
document.addEventListener('DOMContentLoaded', () => {
    // Тема
    const savedTheme = localStorage.getItem('theme') as 'light' | 'dark' | null;
    if (savedTheme) {
        htmlElement.setAttribute('data-theme', savedTheme);
        themeIcon?.setAttribute('data-lucide', savedTheme === 'dark' ? 'sun' : 'moon');
    }

    initIcons();
    handleDoctorModal();
    initBookingFilter();
    initBurger();

    // Мови (візуал)
    document.querySelectorAll('.lang-link').forEach(link => {
        link.addEventListener('click', () => {
            document.querySelectorAll('.lang-link').forEach(l => l.classList.remove('active'));
            link.classList.add('active');
        });
```

```
    });
});

:root {
    /* Light Theme */
    --bg-color: #ffffff;
    --section-bg: #f8fafc;
    --text-main: #1e293b;
    --text-muted: #64748b;
    --accent: #2563eb;
    --accent-hover: #1d4ed8;
    --card-bg: #ffffff;
    --border: #e2e8f0;
    --nav-bg: rgba(255, 255, 255, 0.8);
}

[data-theme="dark"] {
    --bg-color: #0f172a;
    --section-bg: #1e293b;
    --text-main: #f1f5f9;
    --text-muted: #94a3b8;
    --accent: #3b82f6;
    --card-bg: #1e293b;
    --border: #334155;
    --nav-bg: rgba(15, 23, 42, 0.8);
}

html { font-size: 16px; }
@media (max-width: 1500px) { html { font-size: 15px; } }
@media (max-width: 768px) { html { font-size: 14px; } }

img { max-width: 100%; height: auto; }

.container {
    width: 100%;
    padding: 0 1.5rem;
}

* { box-sizing: border-box; margin: 0; padding: 0; font-family: 'Inter', sans-serif; }
body { background: var(--bg-color); color: var(--text-main); transition: 0.3s; line-height: 1.6; }
.container { max-width: 1200px; margin: 0 auto; padding: 0 20px; }

.navbar {
    position: fixed;
    width: 100%;
    top: 0;
    z-index: 1000;
    background: var(--nav-bg);
    backdrop-filter: blur(10px);
    border-bottom: 1px solid var(--border);
    padding: 1rem 0;
```

```
    transition: all 0.3s ease;
}

.navbar .container {
    display: flex;
    justify-content: space-between;
    align-items: center;
}
.logo { font-weight: 800; font-size: 1.5rem; color: var(--text-main); }
.logo span { color: var(--accent); }
.nav-menu a { margin: 0 15px; text-decoration: none; color: var(--text-main); font-weight: 500; font-size: 0.95rem; }
.nav-actions { display: flex; align-items: center; gap: 15px; }

/* Buttons */
.btn-primary { background: var(--accent); color: white; padding: 10px 24px; border-radius: 8px; text-decoration: none; font-weight: 600; transition: 0.2s; border: none; cursor: pointer; }
.btn-primary:hover { background: var(--accent-hover); }
.btn-outline { border: 2px solid var(--accent); color: var(--accent); padding: 8px 22px; border-radius: 8px; text-decoration: none; font-weight: 600; }
.btn-icon { background: none; border: 1px solid var(--border); color: var(--text-main); padding: 8px; border-radius: 8px; cursor: pointer; }

/* Hero */
.hero { padding: 160px 0 100px; background: var(--section-bg); }
.hero-grid { display: grid; grid-template-columns: 1fr 1fr; gap: 60px; align-items: center; }
.hero h1 { font-size: 3.5rem; line-height: 1.1; margin-bottom: 20px; }
.hero p { font-size: 1.2rem; color: var(--text-muted); margin-bottom: 30px; }
.hero-btns { display: flex; gap: 15px; }
.image-placeholder { width: 100%; height: 400px; background: var(--border); border-radius: 20px; }

/* Stats */
.stats-section { padding: 60px 0; border-bottom: 1px solid var(--border); }
.stats-grid { display: grid; grid-template-columns: repeat(3, 1fr); text-align: center; }
.stat-item h3 { font-size: 2.5rem; color: var(--accent); }

/* Carousel (Simplified) */
.carousel-section { padding: 100px 0; }
.section-title { text-align: center; margin-bottom: 50px; font-size: 2rem; }
.carousel-container { display: flex; align-items: center; gap: 20px; position: relative; }
.carousel-track { display: flex; gap: 20px; overflow: hidden; width: 100%; }
.spec-card { min-width: 220px; background: var(--card-bg); padding: 40px 20px; border-radius: 16px; border: 1px solid var(--border); text-align: center; transition: 0.3s; }
.spec-card:hover { transform: translateY(-5px); border-color: var(--accent); }
.spec-card i { color: var(--accent); margin-bottom: 15px; }

/* Doctors */
.doctors-grid { display: grid; grid-template-columns: repeat(3, 1fr); gap: 30px; }
.doctor-card { background: var(--card-bg); border-radius: 16px; overflow: hidden; border: 1px solid var(--border); }
.doc-img { height: 250px; background: var(--border); }
```

```
.doc-info { padding: 25px; }
.doc-info h4 { margin-bottom: 5px; }
.doc-info span { color: var(--accent); font-size: 0.9rem; font-weight: 600; display: block; margin-bottom: 10px; }
.doc-info p { font-size: 0.9rem; color: var(--text-muted); margin-bottom: 15px; }
.btn-text { background: none; border: none; color: var(--accent); font-weight: 600; cursor: pointer; }

/* Footer */
.footer { padding: 80px 0 30px; background: var(--section-bg); border-top: 1px solid var(--border); }
.footer-grid { display: grid; grid-template-columns: 2fr 1fr 1.5fr; gap: 40px; margin-bottom: 50px; }
.footer-links a { display: block; text-decoration: none; color: var(--text-muted); margin-bottom: 10px; }
.footer h5 { margin-bottom: 20px; }
.footer-bottom { text-align: center; border-top: 1px solid var(--border); padding-top: 30px; color: var(--text-muted); font-size: 0.9rem; }

/* Language Links Styling */
.lang-links {
    display: flex;
    align-items: center;
    gap: 8px;
    margin-right: 10px;
    font-size: 0.85rem;
    font-weight: 600;
    letter-spacing: 0.5px;
}

.lang-link {
    text-decoration: none;
    color: var(--text-muted);
    transition: 0.2s;
    padding: 2px 4px;
}

.lang-link:hover {
    color: var(--accent);
}

.lang-link.active {
    color: var(--text-main);
    pointer-events: none; /* Вимикаємо клік на поточну мову */
}

.lang-divider {
    color: var(--border);
    font-weight: 300;
    user-select: none;
}

/* Приховуємо розділювач на мобільних, якщо потрібно буде переносити */
@media (max-width: 768px) {
```

```
    .lang-links {
        margin-right: 5px;
    }
}

.spec-icon-img {
    width: 48px;        /* Фіксована ширина */
    height: 48px;       /* Фіксована висота */
    object-fit: contain; /* Щоб іконка не деформувалася */
    margin-bottom: 20px;
    transition: transform 0.3s ease;

    /* Якщо це чорна іконка SVG, а у нас темна тема: */
    filter: var(--icon-filter);
}

/* Налаштування фільтра для темної теми */
[data-theme="dark"] {
    --icon-filter: brightness(0) invert(1); /* Робить чорну іконку білою */
}

[data-theme="light"] {
    --icon-filter: none;
}

.spec-card:hover .spec-icon-img {
    transform: scale(1.1); /* Легкий ефект при наведенні */
}

/* Hero Section */
.about-hero {
    padding-top: 140px;
    background: var(--bg-color);
}

.hero-label {
    color: var(--accent);
    text-transform: uppercase;
    font-weight: 700;
    font-size: 0.85rem;
    letter-spacing: 2px;
    margin-bottom: 15px;
}

.about-hero h1 {
    font-size: 3.5rem;
    max-width: 800px;
    line-height: 1.1;
    margin-bottom: 60px;
}
```

```
.about-hero-image {
    width: 100%;
    height: 500px;
    overflow: hidden;
}

.about-hero-image img {
    width: 100%;
    height: 100%;
    object-fit: cover;
    display: block;
}

/* Content Layout */
.about-content {
    padding: 100px 0;
}

.narrow-container {
    max-width: 900px; /* Вужчий контейнер для кращого читання тексту */
}

.content-grid {
    display: grid;
    grid-template-columns: 1fr;
    gap: 40px;
}

.content-lead {
    font-size: 1.5rem;
    line-height: 1.4;
    font-weight: 500;
    color: var(--text-main);
}

.content-main p {
    font-size: 1.1rem;
    color: var(--text-muted);
    margin-bottom: 25px;
}

/* Quote style */
blockquote {
    border-left: 4px solid var(--accent);
    padding-left: 30px;
    margin: 40px 0;
    font-style: italic;
    font-size: 1.3rem;
    color: var(--text-main);
}
```

```
/* Dark theme specific adjustments */
[data-theme="dark"] .about-hero-image {
    filter: brightness(0.8) contrast(1.1);
}

/* Breadcrumbs */
.breadcrumbs {
    padding: 120px 0 20px;
    font-size: 0.9rem;
    color: var(--text-muted);
}
.breadcrumbs a { text-decoration: none; color: var(--text-muted); transition: 0.2s; }
.breadcrumbs a:hover { color: var(--accent); }
.breadcrumbs span { margin: 0 10px; }

/* Section Header */
.section-header {
    margin-bottom: 60px;
    max-width: 700px;
}
.section-header .label {
    color: var(--accent);
    font-weight: 700;
    text-transform: uppercase;
    letter-spacing: 1px;
    font-size: 0.8rem;
    margin-bottom: 10px;
}
.section-header h1 { margin-bottom: 20px; }

/* Grid */
.specialties-grid {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(350px, 1fr));
    gap: 40px;
}

<x-layout>
    <nav class="breadcrumbs">
        <div class="container">
            <a href="/">{{ __('site_labels.home') }}</a> <span>/</span> <span class="current">{{
__('site_labels.cabinet') }}</span>
        </div>
    </nav>

    <section class="profile-page">
        <div class="container">
            <div class="profile-layout">
                <aside class="profile-sidebar">
                    <div class="user-card-static">
                        <div class="user-avatar-large">
```

```
                <i data-lucide="user"></i>
              </div>
              <h2>{{ $user->name }} {{ $user->surname }}</h2>

              <div class="user-quick-stats">
                <div class="stat-item">
                  <span class="stat-value">{{ $user->appointments()->count() }}</span>
                  <span class="stat-label">{{ __('site_labels.visits') }}</span>
                </div>
                <div class="stat-item">
                  <span class="stat-value">{{ $user->appointments->count() }}</span>
                  <span class="stat-label">{{ __('site_labels.active') }}</span>
                </div>
              </div>

              <form action="{{ route('auth.logout') }}" method="POST" class="logout-form">
                @csrf
                <button type="submit" class="btn-logout">
                  <i data-lucide="log-out"></i> {{ __('site_labels.logout') }}
                </button>
              </form>
            </div>
          </aside>

          <main class="profile-main">
            <div class="profile-section">
              <header class="section-header-sm">
                <h3>{{ __('site_labels.personal_info') }}</h3>
                <p>{{ __('site_labels.data_used_for') }}</p>
              </header>

              <form action="{{ route('auth.user.update_profile') }}" method="POST"
class="profile-form">
                @csrf
                @method('PUT')

                <div class="form-grid">
                  <div class="form-group">
                    <label>{{ __('site_labels.name') }}</label>
                    <input type="text" name="name" value="{{ $user->name }}" required>
                  </div>
                  <div class="form-group">
                    <label>{{ __('site_labels.surname') }}</label>
                    <input type="text" name="surname" value="{{ $user->surname }}" required>
                  </div>
                  <div class="form-group">
                    <label>{{ __('site_labels.phone') }}</label>
                    <input type="tel" name="phone" value="{{ $user->phone }}" required>
                  </div>
                  <div class="form-group">
                    <label>{{ __('site_labels.birthdate') }}</label>
```

```
                        <input type="date" name="birthdate" value="{{ $user->birthdate }}">
                    </div>
                    <div class="form-group full-width">
                        <label>{{ __('site_labels.address') }}</label>
                        <input type="text" name="address" value="{{ $user->address }}">
                    </div>
                    <div class="form-group full-width">
                        <label>Email</label>
                        <input type="email" name="email" value="{{ $user->email }}">
                    </div>
                </div>

                <button type="submit" class="btn-primary">{{ __('site_labels.save_changes') }}</button>
            </form>
        </div>

        <div class="profile-section mt-40">
            <header class="section-header-sm">
                <h3>{{ __('site_labels.close_appointments') }}</h3>
            </header>
            <div class="appointments-list">
                @if($user->appointments->isNotEmpty())
                    @foreach($user->appointments as $appointment)
                        <div class="appointment-item">
                            <div class="app-date">
                                <span class="day">{{ $appointment->appointment_at->format('d') }}</span>
                                <span class="month">{{ $appointment->appointment_at->translatedFormat('F') }}</span>
                            </div>
                            <div class="app-info">
                                <h5>{{ $appointment->service->title }}</h5>
                                <p>{{ $appointment->doctor->name }} • {{ $appointment->appointment_at->format('h:i') }}</p>
                            </div>
                            <div class="app-status status-pending">{{ __('site_labels.' . $appointment->status) }}</div>
                        </div>
                    @endforeach
                @else
                    <div class="empty-state">
                        <p>{{ __('site_labels.you_have_no_app') }}</p>
                        <a href="/booking" class="btn-link">{{ __('site_labels.book_now') }}</a>
                    </div>
                @endif
            </div>
        </div>
    </main>
  </div>
</div>
```

```
    </section>
</x-layout>

<x-layout>
    <section class="booking-page">
        <div class="container">
            <header class="section-header">
                <div class="label">Запис на прийом</div>
                <h1>Оберіть зручний час для візиту</h1>
                <p>Заповніть форму нижче, і ми надішлемо підтвердження на вашу пошту.</p>
            </header>

            <form action="{{ route('appointments.store') }}" method="POST" class="booking-form">
                @csrf

                <div class="booking-grid">
                    <div class="booking-selection">
                        <div class="booking-step-card">
                            <div class="step-num">1</div>
                            <h3>Деталі візиту</h3>

                            <div class="form-group">
                                <label for="spec_filter">Напрямок лікування</label>
                                <select id="spec_filter">
                                    <option value="all">Усі напрямки</option>
                                    @foreach($specializations as $spec)
                                        <option value="{{ $spec->id }}">{{ $spec->title }}</option>
                                    @endforeach
                                </select>
                            </div>

                            <div class="form-group">
                                <label for="doctor_id">Оберіть лікаря</label>
                                <select name="doctor_id" id="doctor_id" required>
                                    <option value="" disabled selected>Спочатку оберіть напрямок...</option>
                                    @foreach($doctors as $doctor)
                                        <option value="{{ $doctor->id }}" data-spec="{{ $doctor->specialization_id }}">
                                            {{ $doctor->name }}
                                        </option>
                                    @endforeach
                                </select>
                            </div>

                            <div class="form-group">
                                <label for="service_id">Послуга</label>
                                <select name="service_id" id="service_id" required>
                                    <option value="" disabled selected>Оберіть послугу...</option>
                                    @foreach($services as $service)
                                        <option value="{{ $service->id }}" data-spec="{{ $service->specialization_id }}">
```

```
                        {{ $service->title }} — {{ $service->price }} грн
                    </option>
                @endforeach
            </select>
        </div>

        <div class="form-group">
            <label for="appointment_at">Дата та час</label>
            <input type="datetime-local" name="appointment_at" id="appointment_at"
required min="{{ date('Y-m-d\TH:i') }}">
        </div>
    </div>
</div>

<div class="booking-patient-info">
    <div class="booking-step-card">
        <div class="step-num">2</div>
        <h3>Інформація про пацієнта</h3>

        <div class="form-group">
            <label for="patient_name">Ваше ім'я та прізвище</label>
            <input type="text" name="patient_name" id="patient_name"
                value="{{ auth()->check() ? auth()->user()->full_name : '' }}" required
                placeholder="Іван Іванов">
        </div>

        <div class="form-grid-sm">
            <div class="form-group">
                <label for="patient_phone">Телефон</label>
                <input type="tel" name="patient_phone" id="patient_phone"
                    value="{{ auth()->user()->phone ?? '' }}" required
                    placeholder="+380...">
            </div>
            <div class="form-group">
                <label for="patient_email">Email</label>
                <input type="email" name="patient_email" id="patient_email"
                    value="{{ auth()->user()->email ?? '' }}" required
                    placeholder="example@mail.com">
            </div>
        </div>

        <div class="form-group">
            <label for="notes">Додаткові побажання (нотатки)</label>
            <textarea name="notes" id="notes" rows="4" placeholder="Опишіть вашу
проблему або питання до лікаря..."></textarea>
        </div>

        @auth
            <input type="hidden" name="user_id" value="{{ auth()->id() }}">
        @endauth
```

```
                    @if ($errors->any())
                        <div class="alert alert-danger" style="color: #ef4444; margin-bottom: 20px;
font-weight: 600;">
                            <ul>
                                @foreach ($errors->all() as $error)
                                    <li>{{ $error }}</li>
                                @endforeach
                            </ul>
                        </div>
                    @endif

                    <button type="submit" class="btn-primary btn-booking">Підтвердити
запис</button>
                </div>
            </div>
        </div>
    </form>
</div>
</section>
</x-layout>
```

ДОДАТОК В
(обов’язковий)

КЕРІВНИЦТВО КОРИСТУВАЧА

1. Вебдодаток «Вебзастосунок для запису пацієнтів до лікаря з персоналізованим інтерфейсом» призначений для цифровізації взаємодії між пацієнтами та медичною установою, автоматизації розрахунку робочого часу лікарів та миттєвого бронювання візитів. У системі передбачено рольову модель доступу з трьома основними рівнями, що включають публічний доступ для пацієнтів та закриті модулі для лікарів і адміністраторів.

2. Інструкція для пацієнта

Для налаштування комфортного відображення інтерфейсу пацієнту необхідно скористатися перемикачем колірних тем на верхній панелі сайту. Пошук необхідного спеціаліста здійснюється в загальному каталозі медичних фахівців, де для зручності реалізовано панель фільтрації, яка дозволяє миттєво відсортувати картки лікарів за їхньою конкретною спеціалізацією або медичними послугами. Після переходу на картку обраного лікаря користувач отримує доступ до інтерактивного календаря, де після вибору потрібної дати система автоматично генерує та відображає сітку вільних часових слотів. Щоб здійснити запис на прийом, пацієнту достатньо клікнути на вільний часовий слот, який йому підходить, та натиснути кнопку підтвердження. Після успішної обробки запиту сервером система миттєво заблокує цей час для інших користувачів та видасть повідомлення про успішне бронювання візиту.

3. Інструкція для лікаря та адміністратора

Для отримання доступу до закритих модулів системи лікарю або адміністратору необхідно натиснути кнопку входу у правому верхньому куті екрана та ввести свої персональні реєстраційні дані, які включають електронну пошту та пароль. В межах робочого кабінету адміністратор або лікарі можуть здійснювати керування графіками роботи. Для цього в однойменному розділі обирається конкретний спеціаліст, налаштовуються його робочі дні, години

прийому та тривалість надання кожної окремої послуги, при цьому система автоматично проводить валідацію часу та накладок, щоб унеможливити перетин робочих змін. Адміністратор системи також має доступ до панелі моніторингу подій у реальному часі, де відображається повний список актуальних бронювань, детальна інформація про пацієнтів та статус транзакцій запису. За потреби редагування текстового контенту або додавання нової мовної версії сайту, адміністратор використовує вбудований модуль перекладів, внесені зміни в якому миттєво оновлюють JSON-файли локалізації на сервері без перезавантаження та зупинки системи.

ДОДАТОК Г

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Рисунок Г.1 – Слайд 1

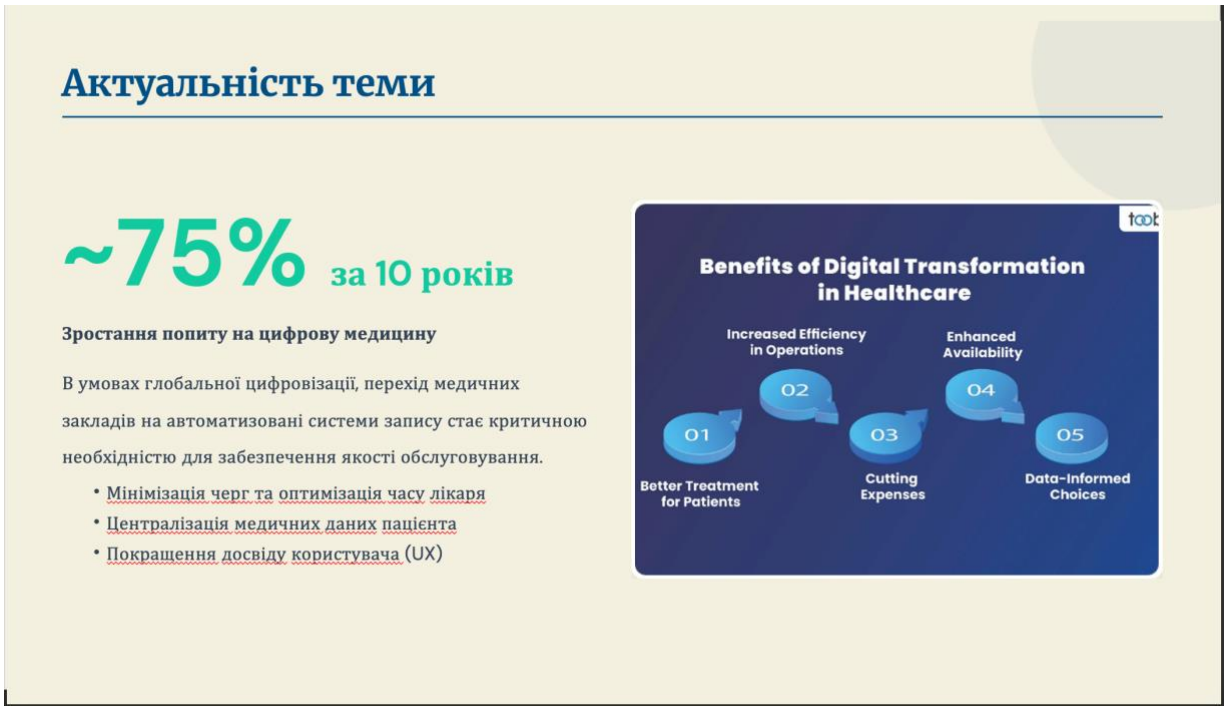


Рисунок Г.2 – Слайд 2

Мета та Завдання

- *Дослідити предметну область та провести порівняльний аналіз існуючих аналогів медичних інформаційних систем.*
- *Спроектувати архітектуру системи та структуру бази даних.*
- *Розробити серверний модуль для керування бізнес-логікою записів, ролями та локалізацією.*
- *Реалізувати інтерактивний інтерфейс.*
- *Провести комплексне тестування програмного забезпечення.*

Рисунок Г.3 – Слайд 3

Аналіз предметної області

Предметна область охоплює процеси цифровізації взаємодії між медичним закладом та пацієнтом для оптимізації управління часовими ресурсами.

- Ієрархічна структура даних: організація взаємозв'язків між спеціалізаціями, лікарями та послугами для забезпечення точного пошуку.
- Динамічне керування розкладом: автоматизація розрахунку вільних слотів на основі тривалості процедур та графіків фахівців.
- Рольова модель доступу (RBAC): суворе розмежування прав користувачів для захисту конфіденційності медичної інформації.
- Консистентність транзакцій: запобігання конфліктам при одночасному бронюванні через механізми атомарності на рівні бази даних.

Рисунок Г.4 – Слайд 4

Аналіз наявного ПЗ (Порівняльна таблиця)

Критерій порівняння	Medics	Health24	Представлений Проєкт
Типізація та безпека коду	Часткова	Середня	**Висока (TypeScript)**
Гнучкість налаштування графіків	Обмежена	Середня	**Повна (Service Layer)**
Швидкість роботи інтерфейсу	Середня	Висока	**Висока**
Локалізація	Статична	Статична	**Динамічна**
Масштаб	Вся країна	Вся країна	**Одна клініка**

Рисунок Г.5 – Слайд 5

Вимоги до програмного забезпечення

Функціональні вимоги

- **Управління записом:** автоматизований пошук лікарів за спеціалізацією та бронювання вільних часових слотів у реальному часі.
- **Адміністрування контенту:** можливість редагування профілів лікарів, налаштування графіків роботи та динамічної локалізації інтерфейсу.
- **Система ролей та доступів:** розмежування прав користувачів (Пацієнт, Лікар, Адміністратор) для доступу до відповідних модулів системи.

Нефункціональні вимоги

- **Надійність та консистентність:** гарантування атомарності операцій бронювання для запобігання конфліктам «подвійного запису».
- **Продуктивність:** час відгуку системи при пошуку та завантаженні графіків не повинен перевищувати 300 мс при середньому навантаженні.
- **Безпека:** захист персональних даних пацієнтів

Рисунок Г.6 – Слайд 6

Архітектура та шаблони проєктування

Model-View-Controller (MVC)

Основа архітектури Laravel, що забезпечує чіткий поділ відповідальності.

Кожен модуль чітко відповідає за свою функціональну частину

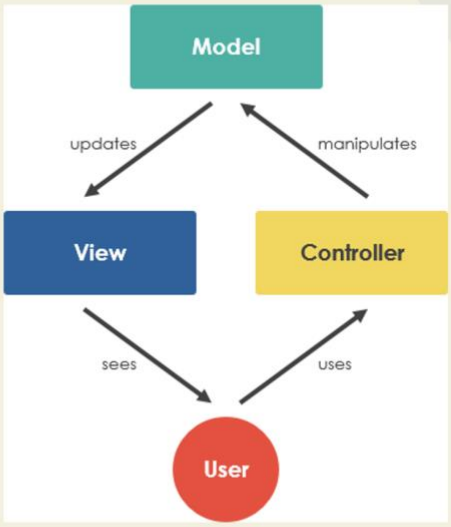
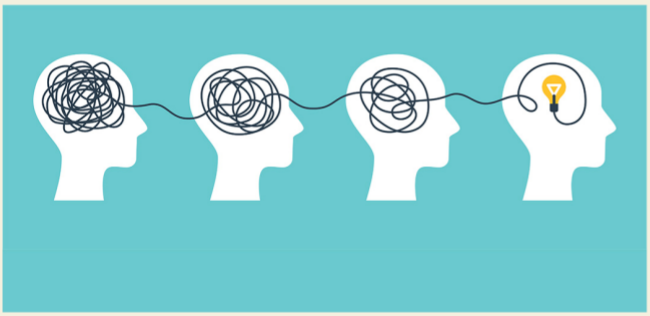


Рисунок Г.7 – Слайд 7

Декомпозиція та інтерфейси



API Layer: Набір REST-ендпоінтів для взаємодії з клієнтом.

Business Logic Layer: Валідація конфліктів часових інтервалів.

Data Layer: Взаємодія з MySQL через Eloquent ORM.

Frontend Layer: Модульна структура TypeScript-компонентів.

Рисунок Г.8 – Слайд 8

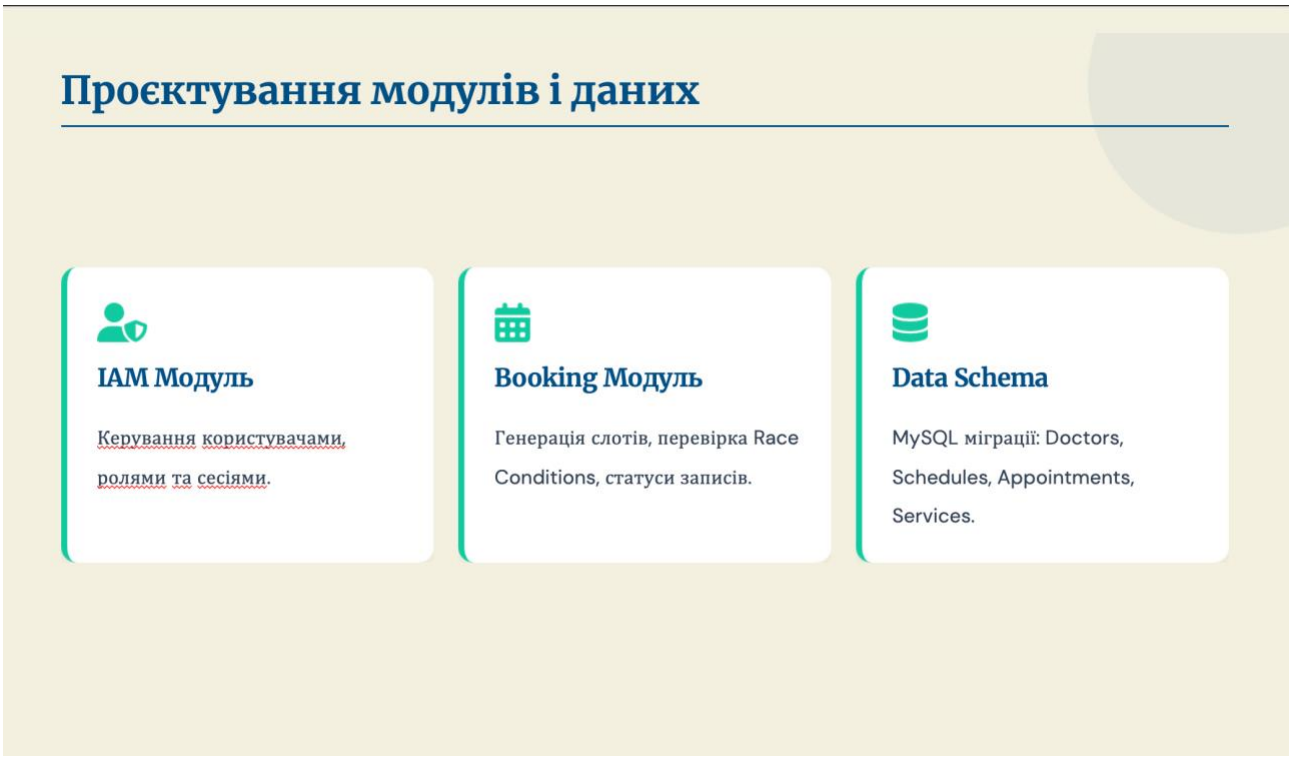


Рисунок Г.9 – Слайд 9

Рисунок Г.10 – Слайд 10

Програмна реалізація та База даних

Реалізовано гнучку схему БД з підтримкою динамічної локалізації та складних графіків.

Використано **Eloquent Factories** для генерації тестових записів для перевірки швидкодії.

Рисунок Г.11 – Слайд 11

Аналіз результатів тестування

Успішно виконано 48 автоматизованих юніт-тестів. Покриття ключової бізнес-логіки сервісного шару становить 100%.

Рисунок Г.12 – Слайд 12

Висновки (Виконання завдань)

Завдання з 3-го слайду	Результат виконання
Дослідження предметної області та аналіз аналогів	Проведено аналіз ринку, визначено ключові недоліки існуючих систем та сформовано вимоги до нового ПЗ.
Проєктування архітектури та структури БД	Розроблено ER-діаграму MySQL та архітектуру на базі Laravel для забезпечення масштабованості системи.
Розробка серверного модуля (бізнес-логіка, ролі, локалізація)	Реалізовано ядро на Laravel: алгоритми розрахунку слотів, користувачі, ролі, доступи та систему динамічної JSON-локалізації.
Реалізація інтерактивного інтерфейсу	Створено адаптивний UI на TypeScript з суворою типізацією, інтерактивними фільтрами та модальними вікнами.

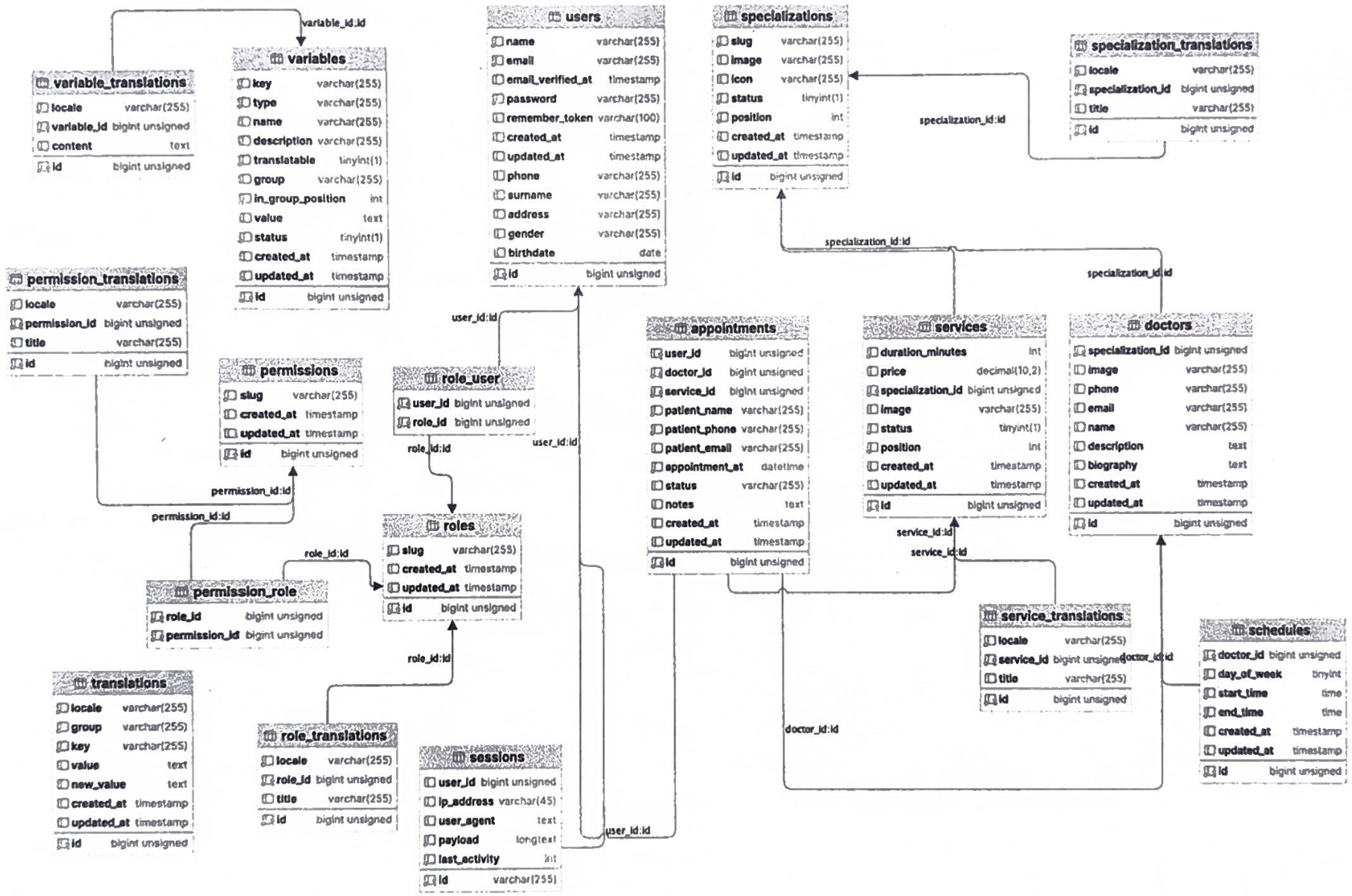
Підсумок: Мета роботи досягнута. Створено масштабовану систему, готову до впровадження.

Рисунок Г.13 – Слайд 13

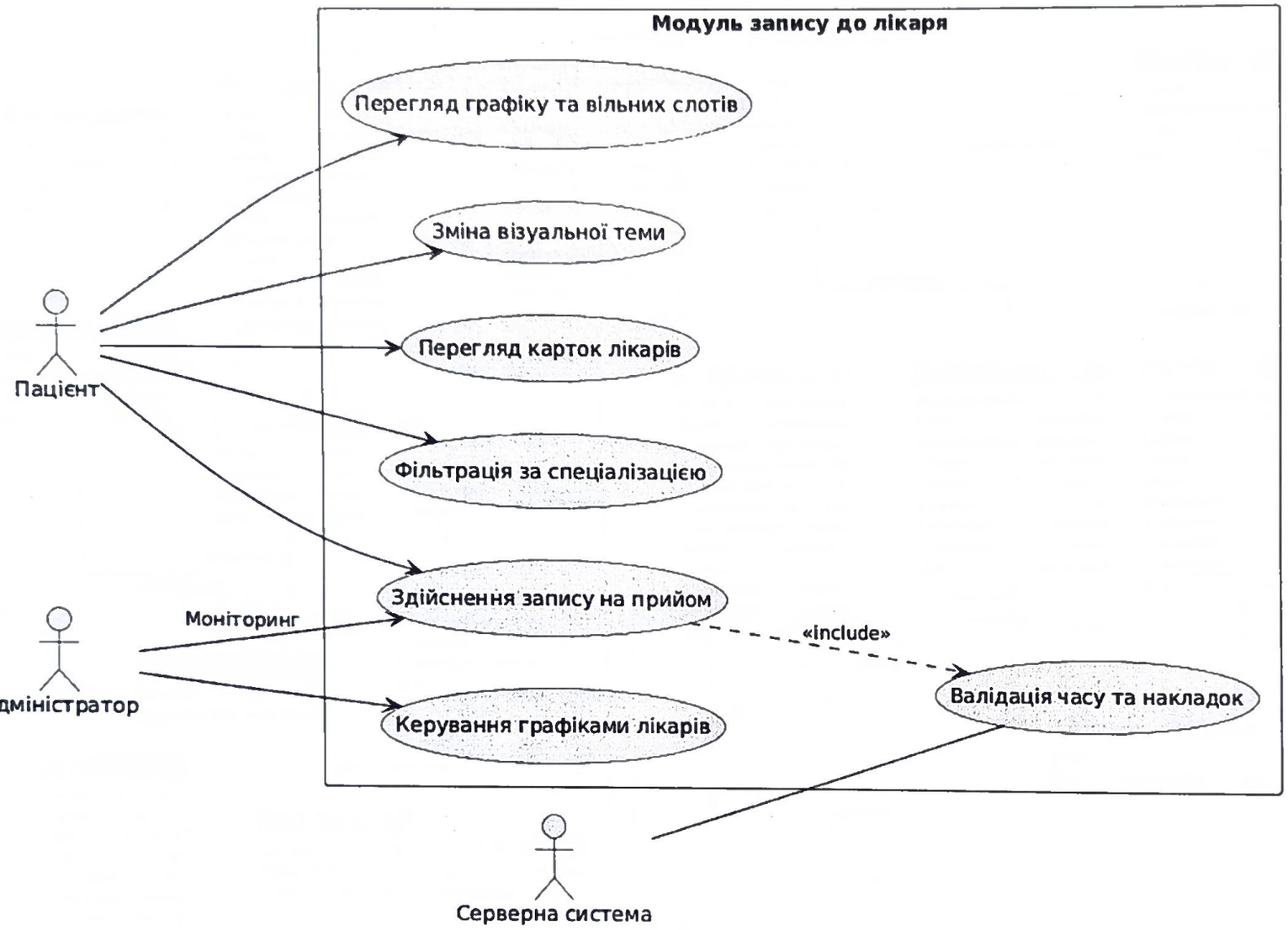
Дякую за увагу!

Рисунок Г.14 – Слайд 14

Графічна частина

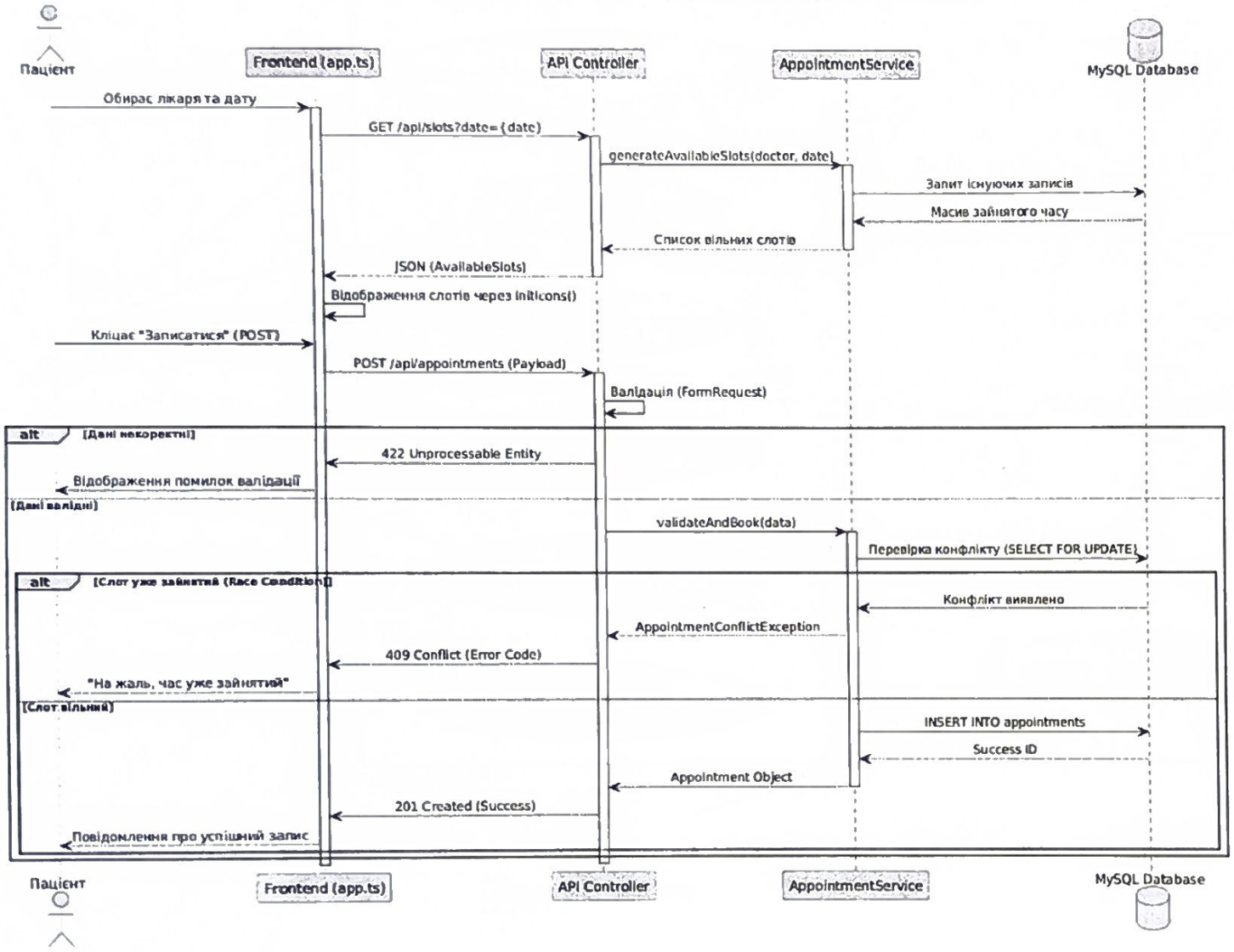


					КвРІПЗс.230134.01.03.Е8			
Змн.	Лист	№ докум.	Підпис	Дата	Діаграма класів	Літ.	Маса	Масштаб
Розробив		Васільєв В. І.						
Керівник		Онишко О. Г.				Арк. 1	Аркушів 3	
Консульт.								
Н. Контр.		Форкун Ю. В.				ХНУ, ІПЗс–23–1		
Затверд.		Бедратюк Л.П.						



					КвРІПЗс.230134.01.03.Е8		
Змн.	Лист	№ докум.	Підпис	Дата			
Розробив		Васільєв В. І.			Діаграма варіантів використання	Літ.	Маса / Масштаб
Керівник		Онишко О. Г.					
Консульт.						Арк. 2	Аркушів 3
Н. Контр.		Форкун Ю. В.				ХНУ. ІПЗс–23–1	
Затверд.		Бедратюк Л.П.					

Діаграма послідовності процесу бронювання візиту



					КвРІПЗс.230134.01.03.Е8		
Змн.	Лист	№ докум.	Підпис	Дата	Діаграма послідовності	Літ.	Маса / Масштаб
Розробив		Васільєв В. І.					
Керівник		Онишко О. Г.				Арк. 3	Аркушів 3
Консульт.							
Н. Контр.		Форкун Ю. В.				ХНУ, ІПЗс–23–1	
Затверд.		Бедратюк Л.П.					

Супровідні документи

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Васільєв Владислав Ігорович

ТемаВебзастосунок для запису пацієнтів до лікаря з персоналізованим інтерфейсом

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3; кількість сторінок записки 65

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі досліджено і проаналізовано предметну область, визначено усі функціональні та нефункціональні вимоги. Був проведений аналіз існуючих програм на ринку, розглянуто їх переваги і недоліки, та доведено актуальність розробки нового програмного забезпечення. Розглянуто інструменти для реалізації спроектованих рішень, в результаті чого створено програмне забезпечення. Також було проведено тестування програми, за результатами якого доведено, що розроблене програмне забезпечення працює коректно та готове до експуатації.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі доведено актуальність теми, визначено мету та завдання дипломного проектування. У першому розділі проведено аналіз предметної області, розглянуто існуючі рішення та визначені функціональні і нефункціональні вимоги до розроблюваного програмного забезпечення. У другому розділі проведено аналіз сучасних архітектур, розглянуто їх переваги і недоліки та визначено, що система буде відповідати монолітній архітектурі та моделі клієнт-сервер. У третьому розділі підготовлено всі залежності для написання коду та виконано практичну розробку програмних модулів і описано їх особливості, в результаті чого створено програмний продукт. Також у цьому розділі виконано модульне тестування системи та проведено його у відповідності до функціональних вимог, в результаті було підтверджено коректну роботу програми.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є актуальною, оскільки на сьогодні в Україні методи запису до лікаря не є достатньо розвинутими та не мають достатньої кількості функціональних можливостей. Також було застосовано новітні технології для побудови програмного продукту та актуальні архітектурні рішеня.

5. Негативні сторони роботи У роботі запис відсутній пошук з фільтрами по лікарях, також немає можливості записатися до одного і того самого лікаря двічі за один прийом.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики проектування. Графічний матеріал дає можливість наочно побачити деталі проектування системи.

8. Інші зауваження _____

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ Канд. технологічних наук. Доцент Кочерука Артем Олександрович.

“____” _____ 2026 р. _____
(підпис)

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Владислав ВАСІЛЬЄВ

Співавтор:

Назва: Вебзастосунок для запису пацієнтів до лікаря з персоналізованим інтерфейсом

Науковий керівник: канд. пед. наук, доцент Оксана ОНИШКО

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:8.21%

Коефіцієнт подібності 2:6.05%

Мікропробіли: 28

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2026-05-20 01:14:31.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата 27.05.26

експерт (Горщук Ю.В.)

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ДЕКЛАРАЦІЯ УЧАСНИКА ОСВІТНЬОГО ПРОЦЕСУ

щодо дотримання академічної доброчесності

Цією декларацією я, Васільєв Владислав Ігорович,

студент III курсу спеціальності 121 – Інженерія програмного забезпечення, група ІПЗс-23-1

здобувач вищої освіти (шифр та назва спец-ті, курс, академічна група)

підтверджую, що ознайомився з Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті та Кодексом академічної доброчесності і **зобов'язуюсь** дотримуватися їх вимог під час освітнього процесу, проведення наукової діяльності, виконання організаційно-адміністративних функцій тощо.

Усвідомлюю, що у разі порушення мною принципів академічної доброчесності нестиму відповідальність перед академічною спільнотою ХНУ згідно з нормами, визначеними Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, законодавства України.

2 вересня 2026 р.

Підпис

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів.
Назва кваліфікаційної роботи: «Вебзастосунок для запису пацієнтів до лікаря з персоналізованим інтерфейсом»
Автор: Васільєв Владислав Ігорович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Онишко Оксана Григорівна, кандидат технічних наук, доцент
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована.	
3	Виявлені запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Виявлені запозичення частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системою перевірки на плагіат Anti-Plagiarism виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках, у структурі змісту, назвах назвах розділів/підрозділів, рамках форм, у назвах та URL-адресах публікацій переліку джерел посилання;

2) запозичення, виявлені у тексті роботи, с фрагментарними.

Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 2.0% з одного джерела. Загальна сумарна подібність у базі даних складає 5% за символами та 8% за лексемами. Крім того, за результатами додаткового аналізу коефіцієнт подібності 1 становить 8.21%, коефіцієнт подібності 2 – 6.05%. Виявлено 28 мікропробілів, зайвих білих знаків або маніпуляцій з інтервалами. З урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 3.06.2026

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Оксана ОНИШКО

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Васільєва Владислава Ігоровича
факультет ІТ, ІІІ курс, група ІПЗ-23-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

25.05.2026.
дата

підпис

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи ІПЗс-23-1
Васільєв В.І.
Прізвище, ініціали

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення»:
Вебзастосунок для запису пацієнтів до лікаря з персоналізованим інтерфейсом

(керівник роботи – Онишко О. Г.)
Прізвище, ім'я, по батькові

25.05.2026
Дата

Підпис студента