МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ГРИБАНІНА АНАСТАСІЯ ОЛЕКСАНДРІВНА

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій,
д-р. техн. наук, професор
_____Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ОБСЛУГОВУВАННЯМ КЛІЄНТІВ САЛОНУ КРАСИ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Надія ПОТАПОВА, доцент кафедри
інформаційних технологій,
к. е. н., доцент

_____

Оцінка: _____ / _____ / _____
(бали за шкалою ЄКТС / за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця - 2026

АНОТАЦІЯ

Грибаніна А. О. Розробка вебзастосунку для управління обслуговуванням клієнтів салону краси. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2026.

У кваліфікаційній (бакалаврській) роботі досліджено проблематику цифровізації бізнес-процесів салонів краси та розроблено повнофункціональний вебзастосунок «Шарм». Система забезпечує онлайн-запис на послуги, управління розкладом майстрів, обробку платежів (з симуляцією), real-time чат між клієнтом і майстром, персональні кабінети для трьох ролей (клієнт, майстер, адміністратор) та аналітичні дашборди. Застосунок побудовано на стекі React + Vite + Tailwind CSS (фронтенд) та Express + SQLite + PostgreSQL + Socket.IO (бекенд), з використанням JWT-аутентифікації, React Query та Zustand для ефективного керування станом.

Ключові слова: вебзастосунок, салон краси, онлайн-запис, управління розкладом, real-time чат, React, Express, SQLite, PostgreSQL.

73 ст., 62 рис., 10 табл., 2 дод., 42 джерела.

ABSTRACT

Hrybanina A.O. Development of a Web Application for Managing Customer Service in a Beauty Salon. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2026.

The qualification (bachelor's) thesis investigates the challenges of digital transformation in beauty salons and develops the full-featured web application «Sharm». The system provides online booking, master schedule management, payment processing (simulated), real-time client-master chat, personalized dashboards for three roles (client, master, administrator) and analytical overviews. The application is built using the React + Vite + Tailwind CSS stack (frontend) and Express + SQLite + PostgreSQL + Socket.IO (backend), with JWT authentication, React Query and Zustand for efficient state management.

Keywords: web application, beauty salon, online booking, schedule management, real-time chat, React, Express, SQLite, PostgreSQL.

ЗМІСТ

ВСТУП

Сучасна beauty-індустрія в Україні активно трансформується під впливом цифрових технологій, що дозволяє салонам краси оптимізувати ключові бізнес-процеси та суттєво покращувати взаємодію з клієнтами. Онлайн-системи управління обслуговуванням клієнтів стають невід'ємною частиною конкурентоспроможності закладів, забезпечуючи автоматизацію записів, персоналізацію пропозицій та оперативну комунікацію. Впровадження веб-застосунків для бронювання послуг сприяє зменшенню адміністративного навантаження на 30–40 %, підвищенню завантаженості майстрів та зростанню лояльності клієнтів завдяки зручному доступу до сервісу 24/7 через будь-який пристрій.

Актуальність теми дослідження. Проблематика ефективного управління обслуговуванням клієнтів у салонах краси полягає в необхідності поєднання високої якості послуг із швидким та зручним процесом запису, що особливо важливо в умовах високої конкуренції та сезонних коливань попиту. Багато закладів досі покладаються на телефонне бронювання або месенджери, що призводить до помилок у розкладах, простоїв фахівців та втрати потенційних клієнтів. Існуючі популярні рішення (Booksy, Fresha, Vagaro та інші) пропонують потужний функціонал, однак часто мають суттєві недоліки для українського ринку: високі комісії, обмежену кастомізацію під локальні платіжні системи, недостатню адаптацію до потреб невеликих та середніх салонів. Таким чином, актуальність роботи зумовлена потребою створення власного веб-застосунку, який би поєднував зручний інтерфейс, реальний час оновлення даних, інтеграцію з популярними платіжними сервісами та повну адаптацію до специфіки українського ринку beauty-послуг.

Мета роботи полягає у розробці вебзастосунку для управління обслуговуванням клієнтів салону краси, який забезпечує комплексну автоматизацію процесів онлайн-запису, управління розкладом майстрів, обробки платежів, комунікації в реальному часі та аналітики діяльності закладу.

Об'єктом дослідження є процеси розробки вебзастосунку для управління обслуговуванням клієнтів салону краси в умовах цифрової трансформації beauty-індустрії.

Предмет дослідження – методи, моделі, архітектурні рішення та програмні засоби реалізації вебзастосунку, що забезпечують ефективне управління записами, розкладами, платежами та комунікацією між клієнтами, майстрами й адміністрацією.

Завданнями дослідження було визначено:

– проаналізувати теоретичні аспекти фунцоінування галузі управління обслуговуванням клієнтів салону краси для вивлення основних елементів, що підлагають автоматизації;

– дослідити існуючі підходи та аналоги систем онлайн-бронювання, виявивши їхні сильні та слабкі сторони;

– розробити архітектуру вебзастосунку, моделі даних та бізнес-процесів салону;

– реалізувати ключові алгоритми обробки записів, управління розкладом та реального часу комунікації;

– виконати програмну реалізацію системи, забезпечити її адаптивний інтерфейс та провести комплексне тестування;

– оцінити отримані результати та обґрунтувати напрями подальшого вдосконалення програмного продукту.

Структура кваліфікаційної роботи. Робота складається зі вступу, трьох основних розділів, висновків, списку використаних посилань та додатків. У першому розділі розглядаються теоретичні засади розробки, аналіз предметної області та огляд існуючих рішень. Другий розділ присвячений проектуванню архітектури, моделям бази даних та алгоритмам функціонування системи. Третій розділ містить опис реалізації, особливості інтерфейсу, заходи забезпечення надійності та результати функціонального й модульного тестування.

Практичне значення роботи полягає у створенні функціонального вебзастосунку, адаптованого до потреб українських салонів краси середнього та

малого сегменту. Розроблена система забезпечує повноцінний онлайн-запис, реальний час оновлення розкладів, симуляцію платежів, чат між клієнтом і майстром, аналітику завантаженості та доходів, що сприяє підвищенню операційної ефективності, зменшенню простоїв та покращенню клієнтського досвіду. Отримані результати можуть бути використані як готове рішення для конкретного салону або як основа для подальшої комерціалізації та адаптації під інші заклади beauty-індустрії.

Апробація результатів дослідження. Основні положення та результати кваліфікаційної роботи були апробовані на наукових конференціях та семінарах:

– IV Всеукраїнська науково-практична конференція «Комп’ютерні технології обробки даних» - 2023 (м. Вінниця, 8 грудня 2023 р.);

– VI Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених «Прикладні інформаційні технології» (м. Вінниця, 22 травня 2025 р.);

– VII Всеукраїнська науково-практична конференція здобувачів вищої освіти та молодих вчених «Прикладні інформаційні технології» (м. Вінниця, 15 травня 2026 р.).

РОЗДІЛ 1
ТЕОРЕТИЧНІ ЗАСАДИ РОЗРОБКИ ВЕБЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ОБСЛУГОВУВАННЯМ КЛІЄНТІВ САЛОНУ КРАСИ

1.1 Аналіз галузі управління обслуговуванням клієнтів салону краси

У сучасному світі beauty-індустрія, зокрема салони краси, переживає значну трансформацію завдяки впровадженню цифрових технологій, які дозволяють оптимізувати процеси обслуговування клієнтів. Розробка вебзастосунку для управління обслуговуванням клієнтів салону краси базується на глибокому розумінні предметної області, яка охоплює комплекс взаємодій між клієнтами, фахівцями та адміністративним персоналом.

Управління обслуговуванням клієнтів у салоні краси включає організацію процесів від первинного контакту з клієнтом до завершення послуги та подальшого аналізу ефективності. Згідно з дослідженнями, ефективне управління в цій сфері сприяє підвищенню лояльності клієнтів на 20-30% за рахунок персоналізованого підходу та автоматизації рутинних задач [1].

Аналіз процесів обслуговуванням клієнтів у салоні краси починається з ідентифікації ключових стейкхолдерів. Клієнти салону краси є основними користувачами, які шукають зручні способи запису на послуги, такі як стрижка, манікюр, фарбування волосся чи масаж, з можливістю вибору часу, фахівця та методу оплати. Фахівці (майстри) відповідають за надання послуг, управління своїм розкладом та комунікацію з клієнтами, тоді як адміністратори контролюють загальну діяльність, включаючи фінансові операції та аналітику. Процес обслуговування клієнтів враховує зовнішні фактори, такі як сезонність попиту (наприклад, збільшення записів перед святами) та конкуренцію на ринку, де салони з онлайн-системами мають перевагу в залученні клієнтів [2].

Одним з центральних аспектів аналізу є вивчення бізнес-процесів салону краси. Ці процеси можна поділити на етапи: планування, виконання та контроль. На етапі планування відбувається формування розкладу роботи майстрів,

враховуючи їхню доступність та тривалість послуг, що вимагає точного розрахунку часу для уникнення простоїв.

Виконання включає безпосереднє обслуговування, де важлива синхронізація між записом клієнта та готовністю фахівця. Контроль передбачає моніторинг платежів, відгуків та ефективності, що дозволяє виявляти вузькі місця, наприклад, низьку завантаженість у певні дні. Дослідження показують, що впровадження вебсистем зменшує час на адміністративні завдання на 40%, дозволяючи персоналу зосередитися на якості послуг [3].

Для ілюстрації взаємозв'язків в управлінні салоном краси корисним є розгляд моделі акторів та їхніх взаємодій, представленої на рис. 1.1, де показано основні ролі та потоки даних у системі управління салоном краси.

Рисунок 1.1 – Модель акторів та випадків використання в управлінні салоном краси

Ця модель демонструє, як клієнт взаємодіє з системою для запису та оплати, майстер керує своїм графіком, а адміністратор має доступ до аналітики. Важливо відзначити, що предметна область передбачає інтеграцію з зовнішніми сервісами, такими як платіжні шлюзи чи календарі, для забезпечення безперервності процесів [4].

Управління салоном краси охоплює вимоги до даних та їхньої структури. У салоні краси накопичується значний обсяг інформації: профілі клієнтів (контактні дані, історія візитів), дані про послуги (назва, тривалість, вартість), розклади майстрів та платежі. Ці дані повинні бути захищеними відповідно до

норм GDPR чи аналогічних стандартів, з акцентом на конфіденційність персональної інформації. Аналіз показує, що неефективне управління даними призводить до втрат у 15-20% доходу через дублювання записів чи помилки в розкладах [5].

Для систематизації ключових елементів предметної області доцільно використовувати табличне подання, як у таблиці 1.1, де наведено основні компоненти управління обслуговуванням клієнтів.

Таблиця 1.1 Основні компоненти управління салоном краси

Компонент	Опис	Вимоги до вебзастосунку
Клієнтські профілі	Збір даних про клієнтів: ім’я, контакти, уподобання	Автоматизована реєстрація та персоналізація
Послуги	Каталог з описами, цінами та тривалістю	Фільтрація, пошук та динамічне оновлення
Розклад	Графік роботи майстрів з урахуванням доступності	Візуалізація календаря та перевірка конфліктів
Платежі	Обробка онлайн- та офлайн-платежів	Інтеграція з платіжними системами
Комунікація	Чат, сповіщення про записи та зміни	Real-time оновлення для миттєвої взаємодії
Аналітика	Звіти про доходи, завантаженість та відгуки	Візуалізація даних у графіках та діаграмах

Ця таблиця ілюструє, як компоненти взаємопов’язані та впливають на вимоги до вебзастосунку, забезпечуючи комплексний підхід до управління.

Управління салоном краси враховує виклики, пов’язані з цифровізацією. Наприклад, салони краси часто стикаються з проблемою інтеграції онлайн- та офлайн-процесів, де клієнти можуть записатися через веб, але підтвердження відбувається телефоном. Аналіз демонструє, що веб-застосунки з мобільною адаптацією підвищують конверсію на 25%, оскільки 70% клієнтів шукають послуги через смартфони [6]. Крім того, важливим є аспект доступності: інтерфейс повинен бути інтуїтивним для різних вікових груп, з урахуванням принципів UX/UI дизайну, таких як мінімалізм та швидке завантаження сторінок.

Ще одним аспектом аналізу є економічна складова. У предметній області

управління обслуговуванням клієнтів салону краси ключовим є оптимізація ресурсів: зменшення простоїв майстрів та максимізація доходу. Дослідження вказують, що автоматизовані системи дозволяють підвищити завантаженість на 30% за рахунок алгоритмів рекомендацій слотів та нагадувань клієнтам [7]. Це підкреслює необхідність інтеграції AI-елементів у вебзастосунок для прогнозування попиту.

Для візуалізації потоків процесів в управлінні салоном краси корисною є діаграма активностей, представлена на рис. 1.2, яка відображає послідовність дій від запиту клієнта до завершення послуги.

Рисунок 1.2 – Діаграма активностей процесу обслуговування клієнта в салоні краси

Ця діаграма ілюструє гілкування процесів залежно від доступності ресурсів, що є критичним для ефективного управління.

Отже, аналіз функціонування процесів управління обслуговуванням клієнтів салону краси розкриває необхідність комплексного підходу до розробки вебзастосунку, який інтегрує процеси планування, виконання та контролю з акцентом на користувацький досвід. Такий застосунок не лише автоматизує рутинні операції, але й сприяє стратегічному розвитку бізнесу, підвищуючи конкурентоспроможність салону в цифрову епоху. Подальша розробка повинна базуватися на принципах гнучкості та масштабовуваності, аби адаптуватися до

змін у beauty-індустрії.

1.2 Огляд існуючих підходів та аналогів програмних рішень

В сучасних умовах цифрової трансформації бізнесу, особливо в сфері послуг, розробка вебзастосунків для управління обслуговуванням клієнтів у салонах краси набуває особливого значення, оскільки дозволяє оптимізувати процеси бронювання, взаємодії та аналітики. Існуючі підходи до створення таких систем ґрунтуються на комбінації клієнт-серверної архітектури, інтеграції баз даних та використання фронтенд- і бекенд-технологій, що забезпечують масштабованість і безпеку.

Одним з ключових аспектів є застосування мікросервісної архітектури, яка дозволяє розділити функціональність на незалежні модулі, такі як автентифікація, управління розкладами та платежі, що полегшує підтримку та розширення системи [8]. Такий підхід сприяє підвищенню ефективності, оскільки кожен модуль може розвиватися окремо, зменшуючи ризики збоїв у всій системі. Крім того, значну увагу приділяють інтеграції реального часу, наприклад, через WebSocket-протоколи, для забезпечення миттєвої синхронізації даних про доступні слоти чи повідомлення в чатах, що є критичним для галузей з динамічним графіком, як салони краси [9].

Інший поширений підхід передбачає використання хмарних сервісів для зберігання даних і обробки платежів, що дозволяє уникнути витрат на власну інфраструктуру та забезпечити високу доступність. Наприклад, інтеграція з API платіжних шлюзів, таких як Stripe чи LiqPay, поєднується з базами даних на кшталт PostgreSQL або SQLite для локальних розгортань, що гарантує конфіденційність клієнтських даних відповідно до стандартів GDPR [10].

У контексті інтерфейсів, сучасні рішення акцентують на адаптивному дизайні з використанням фреймворків на зразок React чи Vue.js, де застосовуються компонентні бібліотеки для створення інтуїтивних форм бронювання та дашбордів. Це сприяє зменшенню навантаження на користувача, адже інтерфейси часто включають елементи, як календарі з візуалізацією вільних

слотів чи рекомендаційні системи на базі простих алгоритмів машинного навчання для персоналізації пропозицій [11]. Водночас, підходи до безпеки фокусуються на токен-базованій автентифікації (JWT) та валідації вхідних даних, щоб запобігти вразливостям, таким як SQL-ін’єкції чи несанкціонований доступ до профілів майстрів.

Переходячи до аналізу конкретних програмних рішень, варто розглянути популярні аналоги, які демонструють різноманітність реалізацій у цій сфері. Одним з провідних є Booksy (рис. 1.3) – платформа, розроблена для малого та середнього бізнесу в індустрії краси, яка пропонує комплексне управління бронюваннями через мобільний додаток та веб-інтерфейс. Booksy інтегрує функції онлайн-запису з автоматичним нагадуванням клієнтам через SMS чи email, а також інструменти для маркетингу, як промо-акції та лояльність. Система підтримує інтеграцію з календарями Google та Apple, що полегшує синхронізацію для майстрів, і включає аналітику продажів з візуалізацією у вигляді графіків. Однак, її недоліком є висока комісія за транзакції (близько 2-3%) та обмежена кастомізація для локальних ринків, таких як український, де потрібна глибока інтеграція з місцевими платіжними системами [12].

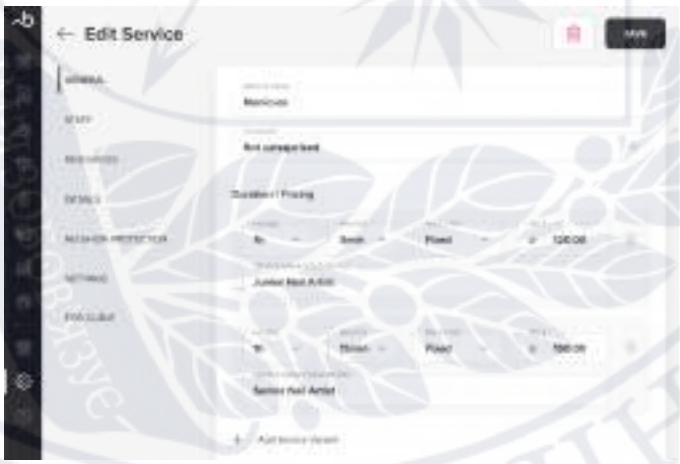

Рисунок 1.3 – Інтерфейс Booksy

Інший аналог – Fresha (рис. 1.4), який позиціонується як безкоштовна система для салонів з акцентом на автоматизацію. Fresha пропонує безлімітне бронювання без комісій, інтеграцію з POS-терміналами для платежів та

інструменти для управління персоналом, включаючи розрахунок зарплат на основі комісій від послуг. Платформа використовує AI для оптимізації розкладів, пропонуючи автоматичні пропозиції слотів на основі історичних даних, і має вбудований чат для комунікації з клієнтами. Серед переваг – мобільна оптимізація та підтримка кількох мов, але вона менш гнучка в частині кастомізації інтерфейсу та вимагає стабільного інтернет-з'єднання для реального часу оновлень [13].

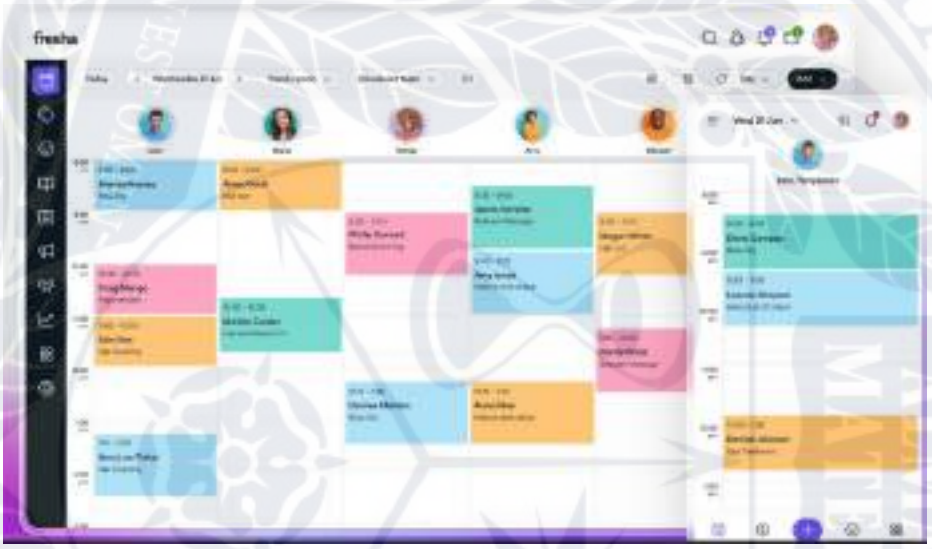

Рисунок 1.4 – Інтерфейс Fresha

Ще одним вартим уваги аналогом є Vagaro (рис. 1.5) – багатофункціональна платформа, орієнтована на салони краси та фітнес-центри, з акцентом на маркетингові інструменти. Vagaro включає онлайн-бронювання з можливістю пакетних послуг (наприклад, комбінація манікюру та педикюру), інтеграцію з соціальними мережами для просування, а також CRM-функції для сегментації клієнтів за частотою візитів. Система підтримує віртуальні платежі через вбудований гаманець та аналітику з дашбордами для моніторингу KPI, як конверсія бронювань. Перевагою є наявність мобільного додатка для майстрів з push-сповіщеннями, але платформа стягує щомісячну абонплату (від 25 доларів) і має обмеження в інтеграції з нестандартними базами даних, що може ускладнити міграцію даних для існуючих салонів [14].

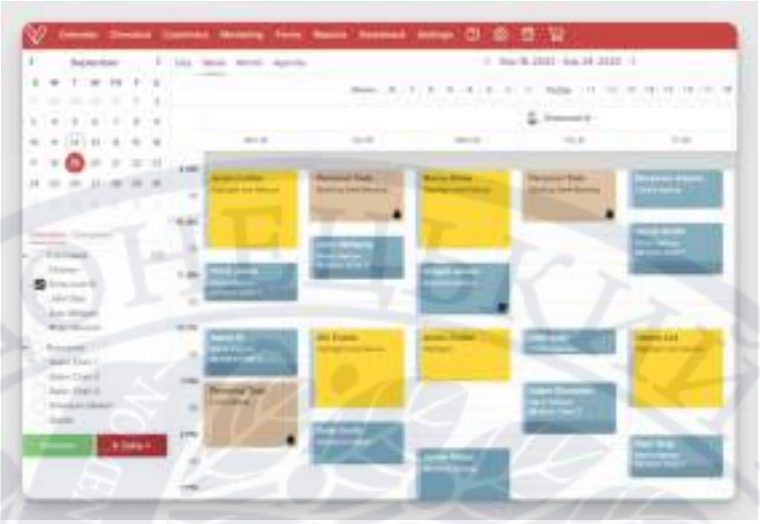

Рисунок 1.5 – Інтерфейс Vagaro

Ці аналоги ілюструють еволюцію від простих систем бронювання до комплексних екосистем, де ключовими є інтеграція з зовнішніми сервісами та фокус на користувацькому досвіді.

Для ілюстрації ключових аспектів цих рішень, наведено схему, яка відображає загальну архітектуру типового вебзастосунку для салонів краси (рис. 1.6).

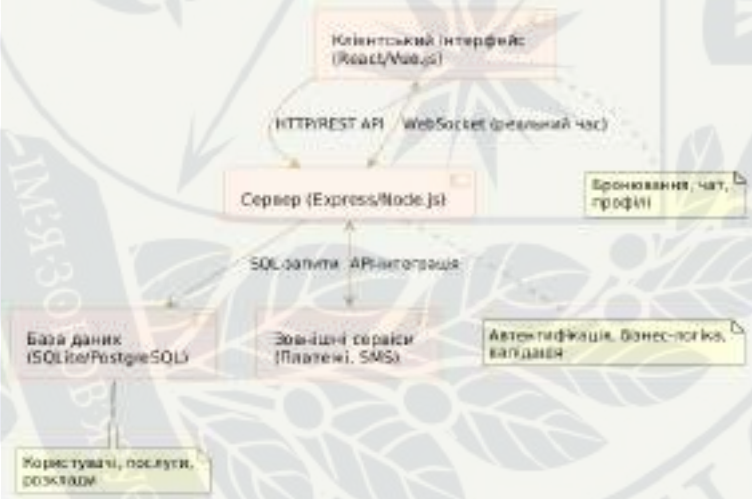

Рисунок 1.6 – Схема архітектури типового вебзастосунку для управління салоном краси

Ця схема демонструє, як компоненти взаємодіють для забезпечення безперервного потоку даних, що є спільним для розглянутих аналогів. Зокрема, Booksy та Fresha активно використовують WebSocket для оновлення розкладів у

реальному часі, тоді як Vagaro акцентує на інтеграції з зовнішніми сервісами для маркетингу.

Порівнюючи ці рішення, можна виділити їхні сильні та слабкі сторони в контексті функціональності, вартості та адаптивності. У наведеній нижче таблиці представлено ключові характеристики аналогів, що дозволяє оцінити їхню ефективність для типового салону краси (табл. 1.2).

Таблиця 1.2 Порівняння характеристик програмних рішень для управління салонами краси

Характеристика	Booksy	Fresha	Vagaro	Розроблений застосунок
Онлайн-бронювання	Повне, з календарем	Автоматизоване, безліміт	Пакетні послуги	Комплексне з рекомендаціями слотів
Інтеграція платежів	З комісією 2-3%	Без комісій	Вбудований гаманець	Симуляція з 95% успіхом
Чат та сповіщення	Базовий чат	Вбудований чат	Push-сповіщення	Real-time через Socket.IO
Аналітика	Графіки продажів	Оптимізація розкладів	KPI-дашборди	Recharts з фільтрами
Вартість	Комісія за транзакції	Безкоштовно	Абонплата від 25$/міс	Безкоштовний для локального використання
Мобільна адаптація	Додаток + веб	Оптимізований мобільний	Додаток для майстрів	Адаптивний дизайн з PWA
Кастомізація	Обмежена	Середня	Висока для маркетингу	Повна, з відкритим кодом

Таблиця 1.2 підкреслює, що кожне рішення має свої акценти: Booksy фокусується на маркетингу, Fresha – на автоматизації без витрат, Vagaro – на CRM, тоді як розроблений застосунок пропонує баланс з акцентом на реальний час та кастомізацію. Загалом, існуючі підходи та аналоги демонструють тенденцію до інтеграції AI та мобільної доступності, що робить їх ефективними інструментами для підвищення конкурентоспроможності салонів краси в цифрову епоху. Подальший розвиток таких систем повинен враховувати локальні регуляції та потреби користувачів для максимальної ефективності.

1.3 Обґрунтування методів дослідження та формування вимог до програмного забезпечення

У процесі розробки вебзастосунку для управління обслуговуванням клієнтів салону краси ключовим етапом є обґрунтування методів дослідження та формування вимог до програмного забезпечення, що забезпечує відповідність системи реальним потребам галузі. Цей підхід ґрунтується на систематичному аналізі теоретичних основ інформатизації бізнес-процесів у сфері послуг, з акцентом на специфіку салонів краси, де поєднуються елементи клієнтського сервісу, автоматизації записів та аналітики даних.

Дослідження в цій області передбачає використання комбінації якісних та кількісних методів, які дозволяють не лише зібрати дані про поточний стан ринку, але й спрогнозувати майбутні тенденції, такі як інтеграція штучного інтелекту для персоналізованих рекомендацій чи мобільної адаптивності для зручності користувачів [15].

Спочатку розглянуто методи дослідження, які обґрунтовуються необхідністю комплексного вивчення предметної області. Аналіз наукової літератури та галузевих звітів є базовим методом, що дає змогу виявити глобальні тренди в цифровізації салонів краси, включаючи перехід до хмарних платформ та використання даних для оптимізації ресурсів. Наприклад, вивчення публікацій про цифрову трансформацію в beauty-індустрії показує, що понад 70% салонів впроваджують онлайн-системи для зменшення простоїв та підвищення лояльності клієнтів, що вимагає глибокого розуміння користувацьких сценаріїв.

Доповнює цей метод емпіричне дослідження через опитування стейкхолдерів – клієнтів, майстрів та адміністраторів салонів, – яке проводиться за допомогою структурованих анкет та інтерв’ю. Такий підхід дозволяє зібрати первинні дані про болісні точки, як-от незручність ручного запису чи відсутність реального часу оновлення розкладів, забезпечуючи об’єктивність через статистичну обробку відповідей. Крім того, застосовується метод порівняльного аналізу існуючих аналогів, таких як платформи для бронювання послуг, де

оцінюються функціональність, інтерфейс та інтеграційні можливості, з метою уникнення типових помилок і адаптації кращих практик до локального контексту [16].

Формування вимог до програмного забезпечення відбувається на основі цих методів і спирається на стандарти, такі як IEEE Std 830-1998, який рекомендує чітке розмежування функціональних та нефункціональних вимог. Функціональні вимоги визначають, що система повинна робити, наприклад, забезпечувати онлайн-реєстрацію, управління розкладами та обробку платежів, тоді як нефункціональні – як вона це робить, охоплюючи аспекти продуктивності, безпеки та usability.

Обґрунтування вибору цих методів полягає в їхній здатності забезпечити ітеративний процес: спочатку формується високорівневий набір вимог через мозковий штурм з експертами, потім деталізується за допомогою use case діаграм, які візуалізують взаємодію акторів з системою. Це дозволяє уникнути перевантаження системи непотрібними функціями та фокусуватися на ключових, як інтеграція з календарями чи аналітика відвідуваності. Важливою є інтеграція методів agile, де вимоги формуються в спринтах з постійним зворотним зв’язком, що підвищує адаптивність до змін у бізнес-середовищі [17].

Для ілюстрації процесу формування вимог розглянуто схему, яка відображає послідовність етапів від збору даних до валідації (рис. 1.7).

Рисунок 1.7 – Схема процесу формування вимог до програмного забезпечення

Схема на рисунку 1.7 демонструє циклічний характер процесу, де кожен етап взаємопов’язаний, забезпечуючи гнучкість у разі виявлення нових вимог, наприклад, щодо мобільної версії додатка. Далі, для конкретизації методів, доцільно систематизувати їх у табличному вигляді, де відображено переваги та застосування в контексті розробки вебзастосунку (табл. 1.3).

Таблиця 1.3 Методи дослідження та їх обґрунтування для формування вимог

Метод дослідження	Обґрунтування	Застосування в проекті
Аналіз літератури	Дозволяє отримати теоретичну базу та уникнути дублювання помилок	Вивчення трендів цифровізації салонів краси
Опитування стейкхолдерів	Забезпечує первинні дані від реальних користувачів	Збір вимог до інтерфейсу та функціональності
Порівняльний аналіз аналогів	Допомагає ідентифікувати кращі практики	Оцінка систем бронювання в beauty-індустрії
Моделювання use case	Візуалізує взаємодії для чіткості вимог	Формування сценаріїв для клієнтів і майстрів
Статистична обробка даних	Підвищує об’єктивність через кількісний аналіз	Аналіз відповідей опитувань для пріоритизації вимог
Ітеративне прототипування	Дозволяє тестувати вимоги на ранніх етапах	Розробка mockup для валідації usability
Експертна оцінка	Забезпечує професійний погляд на вимоги	Консультації з IT-спеціалістами в сфері послуг

Ця таблиця підкреслює, як кожен метод сприяє формуванню всебічних вимог, інтегруючи як теоретичні, так і практичні аспекти. Зокрема, опитування стейкхолдерів обґрунтовується потребою в персоналізованому підході: у салонах краси клієнти часто очікують інтуїтивного інтерфейсу, подібного до мобільних додатків, що підтверджується дослідженнями про зростання мобільного трафіку в e-commerce [18]. Аналогічно, моделювання use case є критичним для визначення акторів – клієнтів, майстрів та адміністраторів – і їхніх ролей, що запобігає конфліктам у вимогах, наприклад, щодо доступу до даних.

Обґрунтування вибору цих методів також пов’язане з економічною ефективністю: аналіз літератури та порівняльний аналіз вимагають мінімальних ресурсів, але дають стратегічний огляд, тоді як опитування та прототипування забезпечують валідацію на практиці. У контексті формування вимог важливо враховувати нефункціональні аспекти, такі як масштабованість системи для зростання кількості користувачів чи забезпечення сумісності з різними пристроями, що обґрунтовується тенденціями до омніканальності в сервісній індустрії [19]. Наприклад, вимога до реального часу оновлень (через WebSocket) випливає з аналізу аналогів, де затримки призводять до подвійних бронювань, а безпека даних – з дотримання GDPR-подібних стандартів для захисту персональної інформації клієнтів.

Додатково, метод експертної оцінки інтегрується для пріоритизації вимог за моделлю MoSCoW (Must-have, Should-have, Could-have, Won’t-have), що дозволяє фокусуватися на критичних функціях, як автоматизоване бронювання, перед вторинними, як персоналізовані рекомендації. Це обґрунтовується дослідженнями про вплив пріоритизації на успіх проєктів, де невизначені вимоги призводять до перевитрат бюджету [20].

Загалом, комбінація цих методів забезпечує баланс між теорією та практикою, гарантуючи, що вебзастосунок не лише задовольняє поточні потреби салону краси, але й є гнучким для майбутніх розширень, таких як інтеграція з AI для аналізу клієнтських уподобань [21].

У підсумку, обґрунтування методів дослідження та формування вимог базується на їхній здатності забезпечити комплексний, ітеративний та користувачо-орієнтований підхід, що є запорукою ефективності програмного забезпечення в динамічній сфері обслуговування клієнтів салонів краси. Такий методологічний фундамент дозволяє мінімізувати ризики та максимізувати цінність системи для всіх стейкхолдерів.

РОЗДІЛ 2
РОЗРОБКА МЕТОДІВ ТА МОДЕЛЕЙ ФУНКЦІОНУВАННЯ ВЕБЗАСТОСУНКУ

2.1 Проєктування архітектури та структури вебзастосунку

Проєктування архітектури та структури вебзастосунку для управління обслуговуванням клієнтів у салоні краси вимагає комплексного підходу, що враховує інтеграцію фронтенд- і бекенд-компонентів, забезпечення безпеки даних та ефективної взаємодії користувачів з різними ролями. У процесі розробки акцент робиться на модульній архітектурі, яка дозволяє легко масштабувати систему та інтегрувати нові функції, такі як онлайн-запис, чат у реальному часі та аналітика.

Основою архітектури є клієнт-серверна модель, де фронтенд реалізовано на базі React з використанням Vite для швидкої збірки, а бекенд – на Express.js з базою даних SQLite для локальної розробки та тестів та PostgreSQL, закладеної в архітектуру, як основної СУБД для серверу, що забезпечує легкість розгортання та низькі витрати на обслуговування [22]. Такий вибір технологій дозволяє досягти високої продуктивності та адаптивності інтерфейсу, адаптованого до різних пристроїв, від мобільних телефонів до десктопів.

Спочатку розглянуто загальну схему архітектури системи, яка ілюструє взаємозв’язки між клієнтською частиною, сервером та зовнішніми сервісами. Рис. 2.1 демонструє високорівневу організацію компонентів, включаючи потоки даних та протоколи комунікації.

Ця архітектура забезпечує розділення обов’язків: фронтенд відповідає за візуалізацію та користувацьку взаємодію, тоді як бекенд обробляє логіку, зберігання даних та інтеграцію з реальним часом через Socket.IO для чату та сповіщень. Важливим аспектом є використання JWT для автентифікації, що дозволяє безпечно керувати сесіями без збереження стану на сервері, а також інтеграцію з cookie для refresh-токенів, що підвищує стійкість до атак [23].

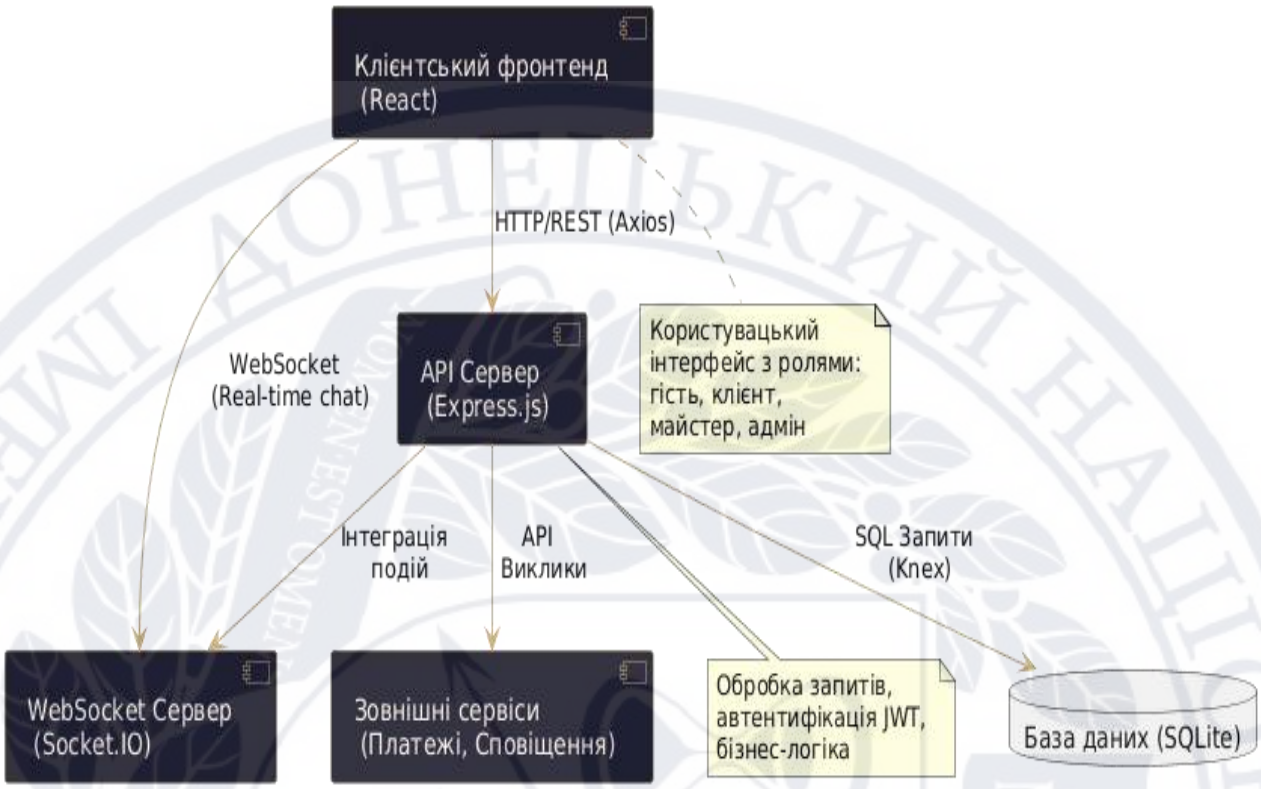

Рисунок 2.1 – Загальна архітектура системи веб-застосунку

Далі доцільно проаналізувати структурну схему системи, яка деталізує ієрархію модулів та їх взаємозв’язки. Структурна схема відображає, як компоненти фронтенду, такі як layouts для різних ролей (ClientLayout, MasterLayout, AdminLayout), інтегруються з публічними сторінками та захищеними маршрутами через React Router. На бекенді структура організована навколо ендпоінтів, як-от /auth, /users, /services, з використанням інтерсепторів Axios для обробки токенів. Рис. 2.2 ілюструє цю організацію, підкреслюючи модульність для полегшення підтримки коду.

Така структура сприяє ефективній розробці, оскільки дозволяє незалежно розвивати фронтенд і бекенд, використовуючи API як інтерфейс. Наприклад, фронтенд використовує Zustand для управління станом автентифікації, що синхронізується з бекендом через Axios, забезпечуючи реактивність інтерфейсу [24]. Це відповідає принципам мікросервісної архітектури, адаптованої для монолітного застосунку, де всі компоненти розгортаються разом для спрощення.

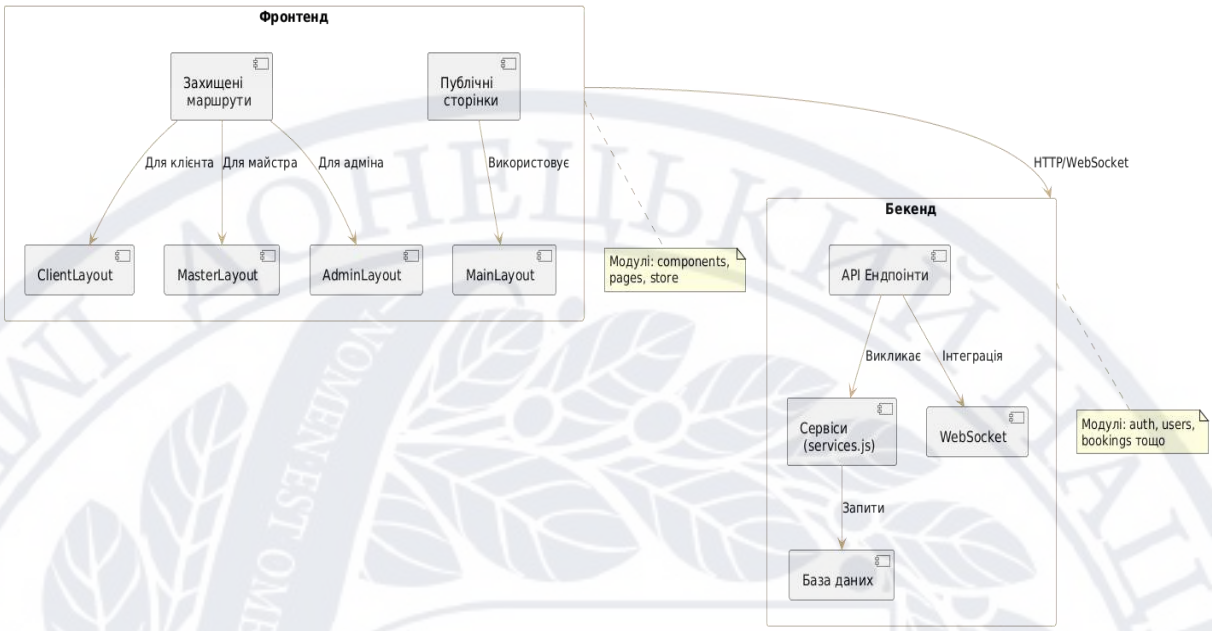

Рисунок 2.2 – Структурна організація модулів системи

Наступним кроком є моделювання взаємодії користувачів з системою через діаграму варіантів використання, яка виділяє ключові сцена.рії для ролей: гість, клієнт, майстер та адміністратор. Діаграма демонструє, як користувачі взаємодіють з функціями, такими як перегляд каталогу, запис на послугу чи управління розкладом, з урахуванням обмежень доступу. Рис. 2.3 показує основні актори та їх взаємодії, підкреслюючи розподіл функціональності.

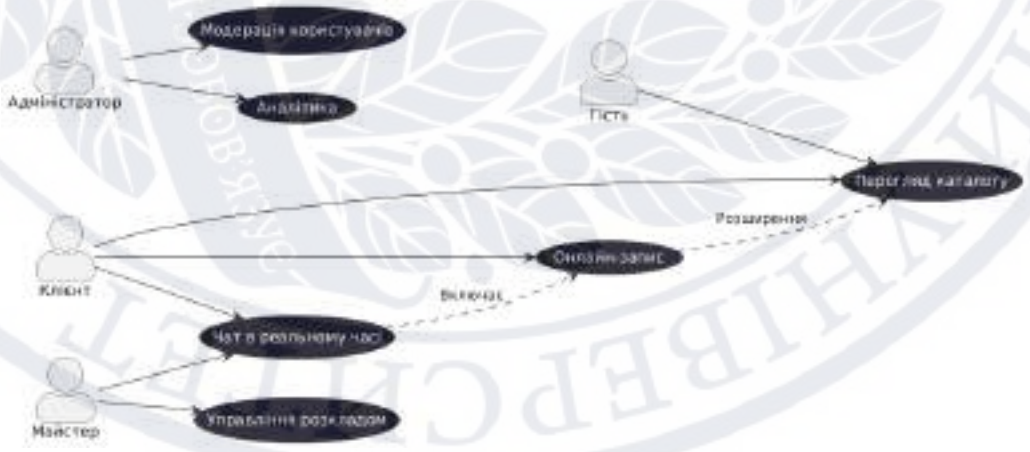

Рисунок 2.3 – Варіанти використання системи для різних ролей

Ця діаграма підкреслює, як базові функції, такі як перегляд, розширюються

для авторизованих користувачів, забезпечуючи гнучкість. Наприклад, клієнт може ініціювати чат лише після запису, що інтегрується з бізнес-логікою на бекенді для перевірки прав [25].

Далі розглянуто діаграму компонентів, яка деталізує внутрішні модулі системи та їх залежності. Компоненти фронтенду, як-от Header, Footer та спеціалізовані сторінки (наприклад, ClientDashboard), взаємодіють з API через сервіси, тоді як бекенд-компоненти обробляють запити. Рис. 2.4 ілюструє цю компонентну структуру, виділяючи інтерфейси та залежності.

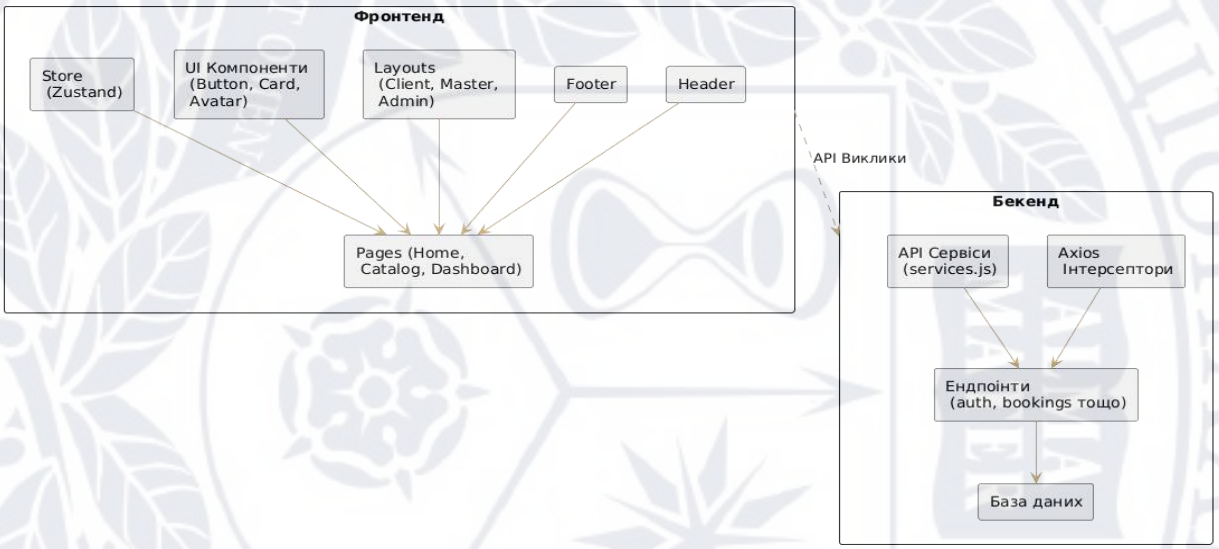

Рисунок 2.4 – Компоненти вебзастосунку та їх залежності

Така компонентна модель полегшує тестування та рефакторинг, оскільки кожен модуль, наприклад, UI-компоненти на базі Tailwind CSS, ізольований і може бути перевикористано [26]. Це також сприяє дотриманню принципів SOLID у проєктуванні.

Для глибшого розуміння об’єктної моделі системи розглянуто діаграму класів, яка моделює ключові сутності, такі як User, Service, Booking, та їх атрибути й методи. Класи відображають структуру бази даних та бізнес-логіку, з асоціаціями між ними. Рис. 2.5 демонструє класову структуру, включаючи спадкування ролей від базового класу User.

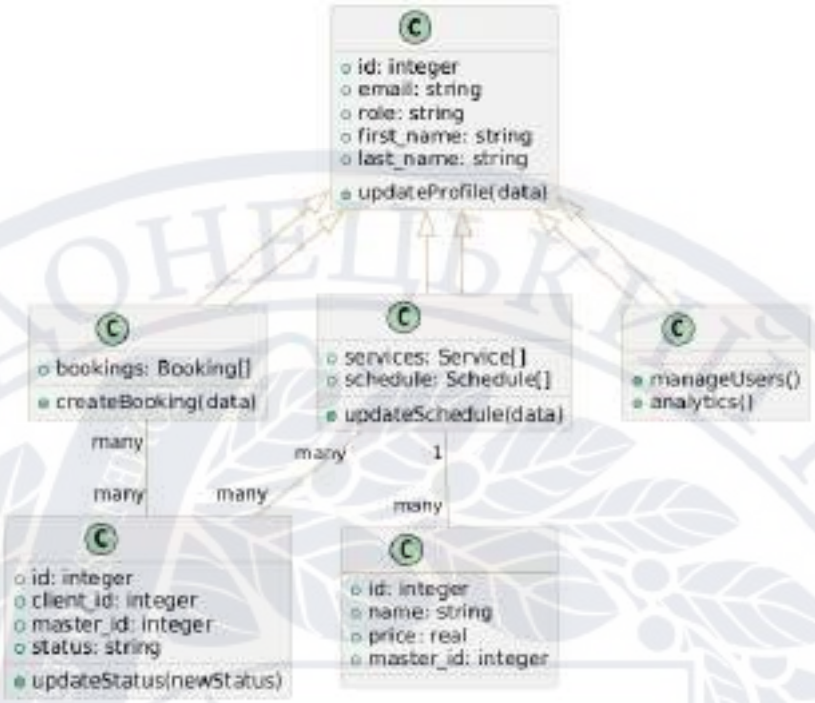

Рисунок 2.5 – Класи системи з атрибутами та методами

Ця діаграма класової моделі забезпечує чітке розуміння даних, наприклад, як Booking пов’язаний з User через ролі, що реалізовано в базі даних з foreign keys [27]. Це дозволяє ефективно керувати станами об’єктів, такими як статус бронювання.

Нарешті, для моделювання динаміки системи розглянуто діаграму станів, яка ілюструє переходи для ключового об’єкта Booking. Стани відображають бізнес-процеси, від створення до завершення чи скасування, з тригерами, такими як підтвердження майстром. Рис. 2.6 показує стани та переходи, підкреслюючи обмеження, як-от часові рамки для скасування.

Початковий стан Pending відображає етап очікування підтвердження після створення запису клієнтом, коли система перевіряє доступність слоту та відсутність конфліктів у розкладі майстра. Перехід до Confirmed відбувається після схвалення майстром і супроводжується автоматичним сповіщенням клієнта, а також блокуванням відповідних часових слотів. Зі стану Confirmed можливі три основні переходи: до InProgress при початку надання послуги, до Cancelled у разі скасування за умови дотримання двогодинного обмеження, або до NoShow при неявці клієнта після закінчення призначеного часу.

Завершальний стан Completed фіксує успішне виконання послуги, після чого система пропонує клієнту залишити відгук, забезпечуючи повний життєвий цикл бронювання з чітко визначеними бізнес-правилами та автоматичними тригерами.

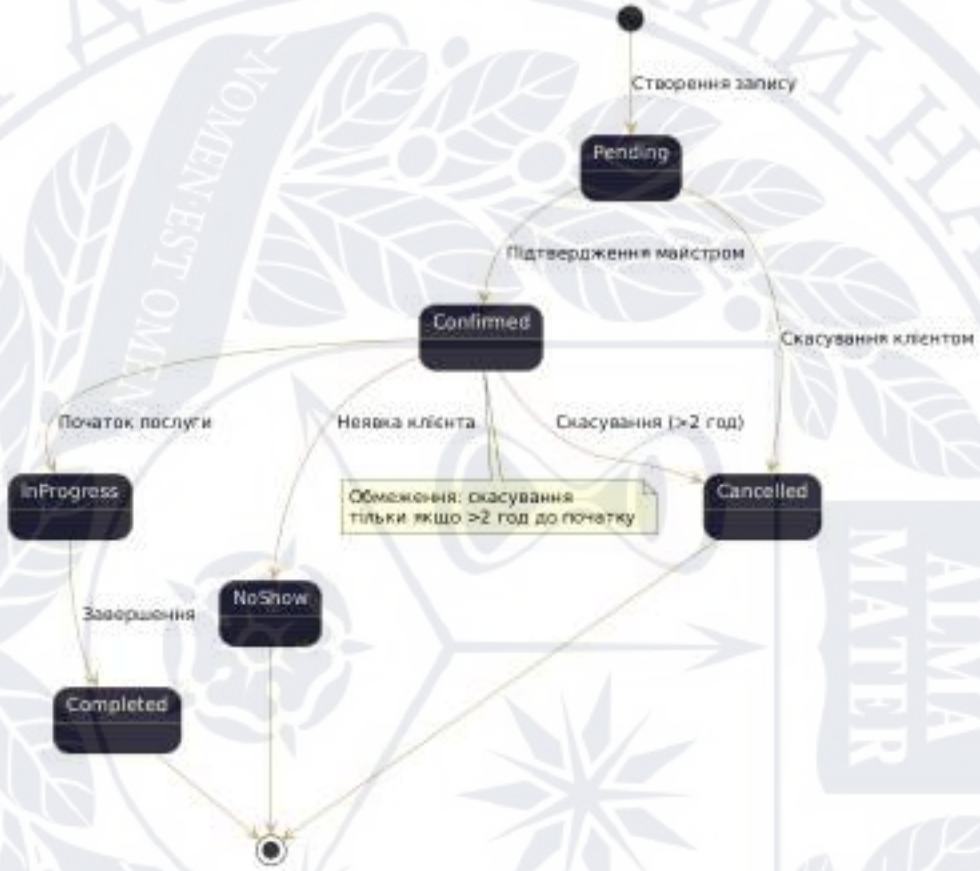

Рисунок 2.6 – Стани об’єкта бронювання з переходами

Ця діаграма станів інтегрується з бізнес-логікою на бекенді, де статуси оновлюються через PATCH-запити, забезпечуючи консистентність даних. Загалом, проєктування архітектури та структури системи базується на принципах модульності, безпеки та користувацької орієнтованості, що дозволяє ефективно управляти обслуговуванням клієнтів у салоні краси, з потенціалом для подальшого розширення, наприклад, інтеграції з мобільними додатками чи AI-рекомендаціями [28]. Такий підхід не тільки оптимізує продуктивність, але й підвищує задоволеність користувачів через інтуїтивний інтерфейс та швидку обробку запитів.

2.2 Розробка алгоритмів обробки даних та управління записами клієнтів

У контексті цифрової трансформації сервісних галузей, зокрема салонів краси, алгоритми обробки даних відіграють визначальну роль у забезпеченні безперебійного управління клієнтськими записами, що дозволяє не тільки оптимізувати операційні процеси, але й створювати персоналізований досвід для кожного відвідувача [29]. У розробленій вебплатформі «Шарм» ці алгоритми інтегровані в єдину систему, де кожний крок – від ініціації бронювання до завершення послуги – підтримується автоматизованими механізмами, що базуються на реальному часі даних та ролевих перевірках. Це дає змогу майстрам фокусуватися на творчій частині роботи, а клієнтам – відчувати себе частиною зручної екосистеми, де технології працюють непомітно, але ефективно, сприяючи лояльності та повторним візитам.

Центральним елементом системи є алгоритм створення запису, який забезпечує швидку та безпечну обробку запиту клієнта (рис. 2.7).

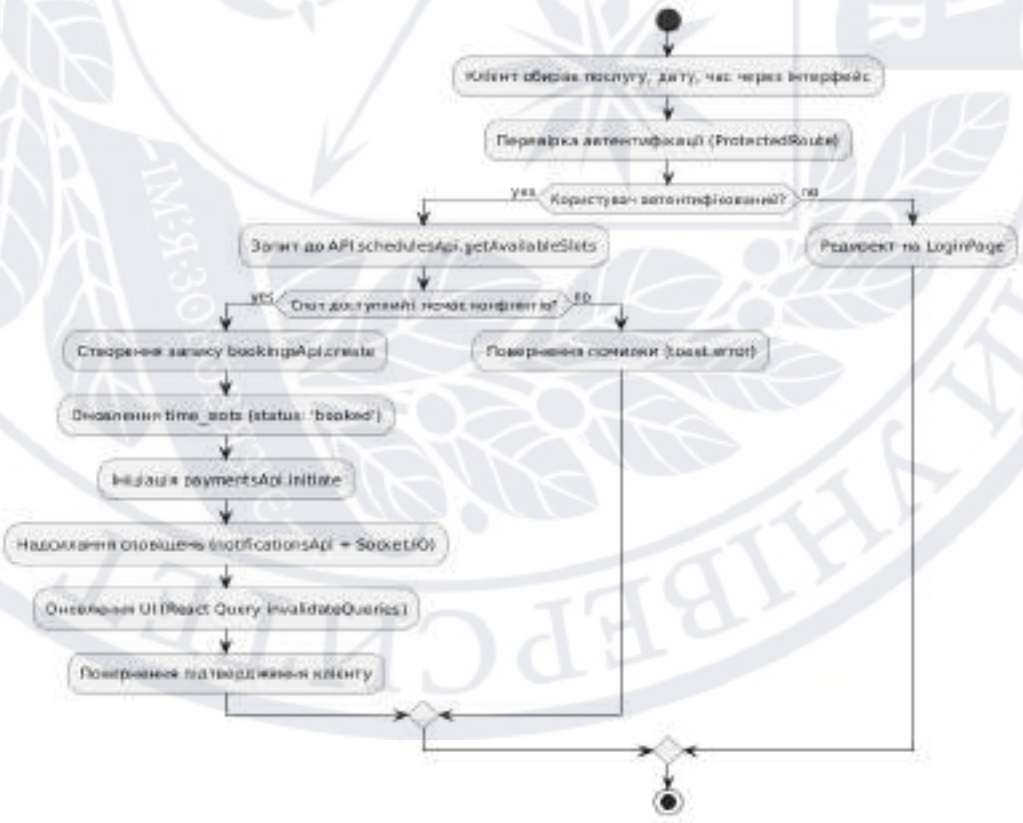

Рисунок 2.7 – Схема алгоритму створення запису клієнта

Як видно з рис. 2.7, процес починається з вибору послуги, дати та часу через клієнтський інтерфейс, після чого система автоматично перевіряє доступність слотів у розкладі майстра за допомогою API-запиту до /schedules/master/:id/slots. Ця перевірка враховує тривалість послуги (duration_minutes), щоб уникнути перетинань, і доповнюється валідацією на рівні клієнта – чи не має він активних записів у той самий часовий проміжок. У разі успіху запис фіксується в базі даних через POST-запит до /bookings, з автоматичним оновленням статусу на ‘pending’ і генерацією пов’язаного платежу. Такий підхід не тільки мінімізує помилки, але й інтегрується з Socket.IO для миттєвого сповіщення майстра, роблячи взаємодію живою та оперативною [30].

Після створення запису алгоритм переходить до динамічного управління його статусом, що є критичним для підтримання балансу між очікуваннями клієнтів і робочим навантаженням майстрів. Цей процес, ілюстрований на Рис. 2.8, передбачає ролеві перевірки: клієнт може ініціювати скасування лише за умови, що до початку послуги залишилося понад дві години, використовуючи DELETE-запит до /bookings/:id з перевіркою canCancelBooking у utils.js. Майстер, у свою чергу, має доступ до PATCH /bookings/:id/status для переходу між станами – від ‘confirmed’ до ‘in_progress’ чи ‘completed’, з обов’язковим оновленням пов’язаних даних, таких як payments і reviews. Адміністратор, завдяки AdminBookingsPage, може втручатися в будь-який момент, забезпечуючи гнучкість у критичних ситуаціях. Інтеграція з cron-задачами в cron.js дозволяє автоматично обробляти прострочені записи: наприклад, скасування неоплачених через 30 хвилин або позначення ‘no_show’ після години запізнення, що звільняє ресурси без ручного втручання [31].

Обробка платежів тісно пов’язана з управлінням записами і реалізується через спеціалізований алгоритм, що забезпечує безпеку та прозорість транзакцій. У AdminPaymentsPage та ClientPaymentPage система використовує paymentsApi для ініціації платежу після створення запису, з симуляцією методів ‘card’ або ‘cash’. Для карткових платежів алгоритм включає генерацію тестових даних

(автозаповнення) і таймер обробки (setTimeout у мутаціях), після чого статус оновлюється на ‘success’ або ‘failed’, автоматично підтверджуючи запис. У разі рефандів (POST /payments/:id/refund) адміністратор може ініціювати повернення, що тригерить оновлення статусу в bookings. Це не тільки захищає від шахрайства, але й інтегрується з аналітикою в AdminDashboard, де revenueData агрегує дані для графіків [32].

Рисунок 2.8 – Схема алгоритму оновлення статусу запису

Таблиця 2.1 демонструє ключові статуси платежів та їх вплив на записи, підкреслюючи, як алгоритми забезпечують узгодженість даних. Як видно з Таблиці 2.1, ці статуси формують ланцюг, де кожен крок алгоритму впливає на наступний, забезпечуючи цілісність даних у реальному часі.

Для підтримки розкладу майстрів алгоритми обробки даних включають bulk-операції в schedulesApi.setWorkDays, де POST /schedules/bulk генерує time_slots на основі work_schedules, з перевіркою на overlaps. Це дозволяє

майстрам швидко налаштовувати графік через MasterSchedulePage, а клієнтам – бачити актуальні слоти в PublicMasterProfilePage. У AdminBookingsPage алгоритм фільтрації та експорту (exportToCSV) агрегує дані з bookings, payments і users, дозволяючи аналізувати тренди, такі як кількість pending-записів чи revenue [33].

Таблиця 2.1 Статуси платежів та відповідні дії в системі управління записами

Статус платежу	Опис процесу	Вплив на запис клієнта
pending	Очікування ініціації оплати після бронювання	Запис у статусі ‘pending’, автоматичне скасування через 30 хв
processing	Симуляція обробки транзакції (2–3 сек)	Тимчасова блокада слоту, сповіщення клієнту
success	Успішна оплата	Автоматичний перехід запису на ‘confirmed’, надсилання підтвердження
refunded	Повернення коштів адміністратором	Оновлення статусу на ‘cancelled’ з нотаткою, рефанд у payments
failed	Невдала транзакція	Скасування запису, сповіщення про помилку

Рис. 2.9 ілюструє алгоритм cron-задач, що автоматизує рутинні операції, звільняючи персонал від монотонної роботи. Алгоритм cron-задач працює як фоновий механізм автоматичного управління записами в реальному часі. При запуску серверу node-cron ініціює чотири паралельні гілки (fork), кожна з яких виконується за своїм розкладом. Перша гілка щогодини перевіряє підтверджені записи, що наближаються (за 24 та 1 годину), формує персональні нагадування та миттєво надсилає їх через Socket.IO. Друга гілка кожні 5 хвилин автоматично скасовує неоплачені записи, створені понад 30 хвилин тому. Третя гілка кожні 30 хвилин переводить прострочені підтверджені записи у статус «no_show». Четверта гілка щодня о 10:00 шукає завершені напередодні записи без відгуків і надсилає клієнтам запрошення залишити відгук. Усі дії супроводжуються детальним логуванням через winston, що забезпечує прозорість і можливість моніторингу роботи вебзастосунку.

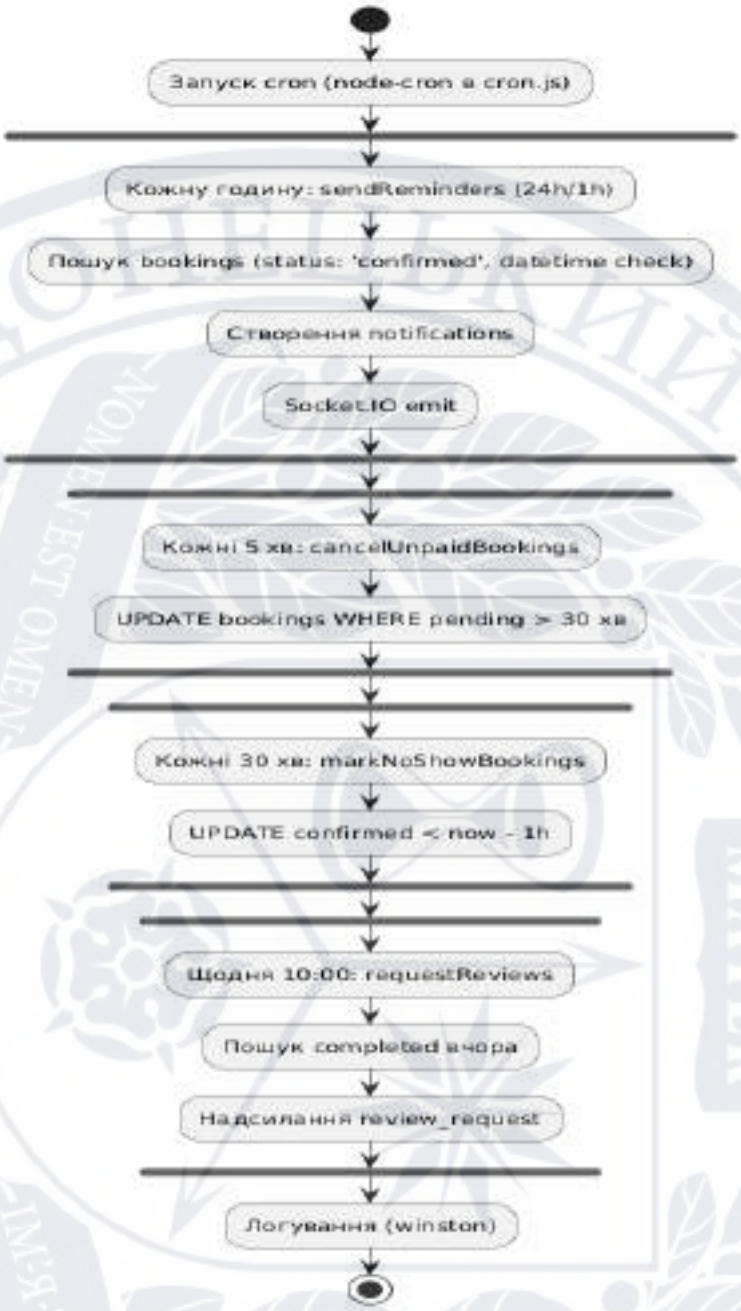

Рисунок 2.9 – Схема алгоритму автоматичної обробки записів cron-задачами

У чаті (chat.socket.js) алгоритми реального часу доповнюють управління записами: при joinConversation перевіряється участь у booking, а send повідомлень оновлює last_message_at. Це створює повноцінну екосистему, де дані обробляються синхронно, а користувачі відчувають миттєву віддачу [34].

Таблиця 2.2 порівнює алгоритми для різних ролей, підкреслюючи їх адаптивність. Як видно з Таблиці 2.2, алгоритми адаптовані до потреб ролей, що робить систему інтуїтивною та масштабовною.

Таблиця 2.2 – Порівняння алгоритмів управління записами за ролями користувачів

Роль	Алгоритм створення	Алгоритм статусів	Алгоритм платежів
Клієнт	Вибір + валідація слотів	Скасування з таймером	Ініціація + симуляція
Майстер	Перегляд доступних	Зміна на in_progress/completed	Перегляд refund
Адмін	Bulk-створення	Повний контроль	Рефанди + аналітика

Загалом, розроблені алгоритми обробки даних у «Шарм» не тільки відповідають сучасним стандартам веброзробки, але й гуманізують взаємодію, перетворюючи технічні процеси на інструмент для створення теплих, персональних відносин між салоном і клієнтами. Вони забезпечують високу ефективність, зменшуючи час очікування та помилки, що в кінцевому підсумку сприяє зростанню бізнесу в конкурентному середовищі цифрових послуг [35].

2.3 Моделювання бази даних та бізнес процесів салону краси

У процесі розробки вебзастосунку для управління обслуговуванням клієнтів у салоні краси «Шарм» особлива увага приділяється моделюванню бази даних та бізнес-процесів, оскільки ці елементи забезпечують ефективну інтеграцію функціональних модулів, таких як онлайн-запис, чат у реальному часі та аналітика. Моделювання бази даних базується на реляційній моделі, яка дозволяє оптимізувати зберігання та доступ до інформації про користувачів, послуги, записи та платежі, враховуючи специфіку салонного бізнесу, де ключовими є часові слоти, ролі користувачів та фінансові транзакції [36].

Використання SQLite як бази для локальної розробки та тестів та PostgreSQL, як основної СУБД для серверу у цьому проєкті обумовлено її легкістю та достатньою продуктивністю для малого та середнього бізнесу, що підтверджується сучасними тенденціями в розробці веб-додатків для сервісних підприємств [37]. Бізнес-процеси, у свою чергу, моделюються з акцентом на автоматизацію взаємодій між клієнтами, майстрами та адміністраторами, забезпечуючи безперервний цикл від бронювання до оцінки послуги.

Моделювання бази даних починається з визначення ключових сутностей та їхніх взаємозв’язків. У проєкті «Шарм» база даних складається з 13 основних таблиць, які охоплюють аспекти аутентифікації, управління послугами, розкладами та комунікаціями.

Центральною сутністю є таблиця users, яка зберігає інформацію про всіх користувачів, включаючи їхні ролі (client, master, admin), що дозволяє реалізувати рольовий доступ до функціоналу. Ця таблиця пов’язана з іншими через зовнішні ключі, забезпечуючи цілісність даних. Наприклад, майстри мають додаткову таблицю master_profiles для зберігання біографічної інформації, досвіду та портфоліо, що інтегрується з послугами через master_id. Таке моделювання дозволяє уникнути дублювання даних і полегшує запити для генерації профілів майстрів у клієнтському інтерфейсі [38].

Для візуалізації структури бази даних розроблено ER-діаграму, яка ілюструє основні таблиці та їхні зв’язки. Рис. 2.10 демонструє ключові сутності бази даних, включаючи користувачів, послуги та записи, з акцентом на зовнішні ключі та кардинальність відносин.

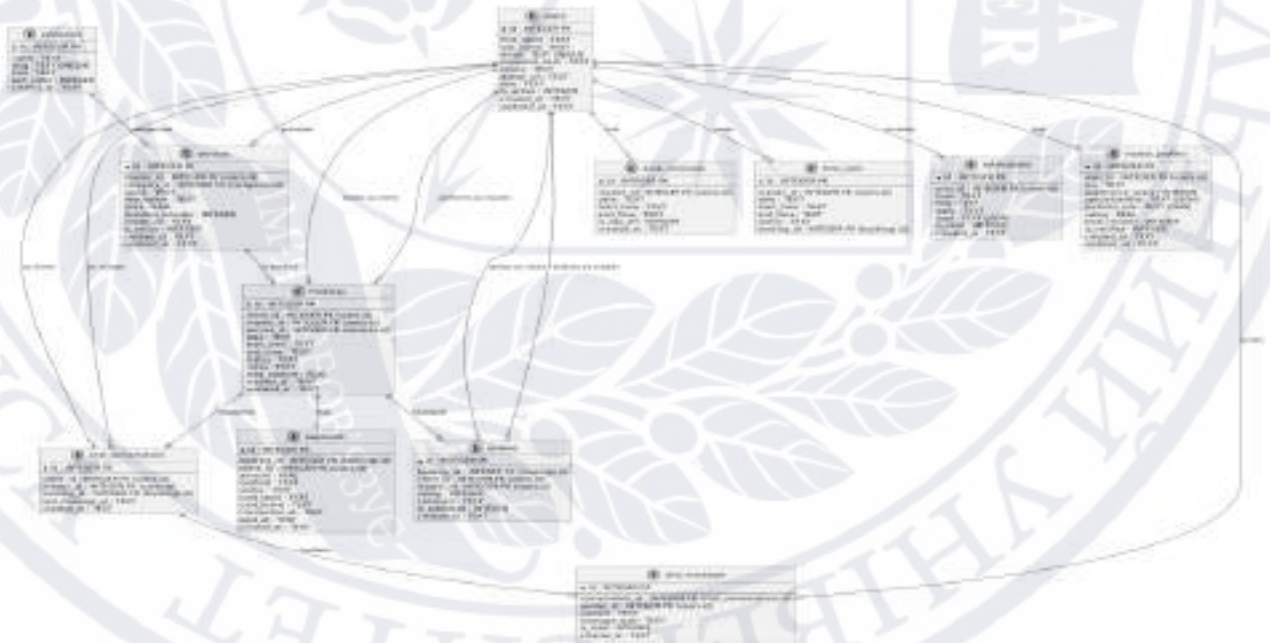

Рисунок 2.10 – ER-діаграма бази даних вебзастосунку «Шарм»

Ця діаграма відображає, як таблиці пов’язані через зовнішні ключі, наприклад, bookings залежить від users (client_id та master_id), services та

time_slots, що забезпечує контроль за доступністю слотів. У моделі передбачено унікальні обмеження, такі як UNIQUE (master_id, date) у work_schedules, щоб уникнути дублювання розкладів, та каскадне видалення для refresh_tokens при видаленні користувача, що підвищує цілісність даних [39].

Детальний опис структури бази даних наведено в таблиці, де вказані ключові поля та їхні призначення. Таблиця 2.3 узагальнює основні таблиці бази даних, включаючи типи даних та обмеження.

Таблиця 2.3 Структура основних таблиць бази даних

Таблиця	Ключові поля	Типи даних	Обмеження та призначення
1	2	3	4
users	id, email, role	INTEGER PK, TEXT UNIQUE, TEXT NOT NULL	Центральна таблиця для аутентифікації; роль визначає доступ (client, master, admin)
master_profiles	user_id, bio, rating	INTEGER FK, TEXT, REAL	Додаткові дані для майстрів; rating обчислюється на основі відгуків
categories	id, name, slug	INTEGER PK, TEXT NOT NULL, TEXT UNIQUE	Категорії послуг для фільтрації в каталозі
services	id, master_id, price, duration_minutes	INTEGER PK, INTEGER FK, REAL NOT NULL, INTEGER NOT NULL	Послуги з ціною та тривалістю; пов’язані з майстрами
work_schedules	master_id, date, start_time, end_time	INTEGER FK, TEXT NOT NULL, TEXT NOT NULL, TEXT NOT NULL	Робочі дні майстрів; UNIQUE (master_id, date)
time_slots	master_id, date, status, booking_id	INTEGER FK, TEXT NOT NULL, TEXT DEFAULT ‘available’, INTEGER FK	Часові слоти; статус: available, booked, blocked

Продовження таблиці 2.3

1	2	3	4
bookings	id, client_id, service_id, status	INTEGER PK, INTEGER FK, INTEGER FK, TEXT DEFAULT ‘pending’	Записи; статуси: pending, confirmed тощо
payments	booking_id, amount, method, status	INTEGER FK UNIQUE, REAL NOT NULL, TEXT NOT NULL, TEXT DEFAULT ‘pending’	Платежі; методи: card, cash
chat_conversations	id, client_id, master_id	INTEGER PK, INTEGER FK, INTEGER FK	Розмови; UNIQUE (client_id, master_id)
chat_messages	conversation_id, sender_id, content	INTEGER FK, INTEGER FK, TEXT NOT NULL	Повідомлення; ON DELETE CASCADE
reviews	booking_id, rating, is_published	INTEGER FK UNIQUE, INTEGER NOT NULL, INTEGER DEFAULT 1	Відгуки; публікація контролюється адміном
notifications	user_id, type, is_read	INTEGER FK, TEXT NOT NULL, INTEGER DEFAULT 0	Сповіщення; типи: booking_confirmed тощо

Ця структура забезпечує ефективне зберігання даних, де, наприклад, duration_minutes у services використовується для розрахунку end_time у bookings, а status у time_slots блокує слоти при бронюванні. Моделювання враховує нормалізацію до 3NF, зменшуючи дублювання, наприклад, спеціалізації зберігаються як JSON у master_profiles для гнучкості [40].

Переходячи до моделювання бізнес-процесів, у салоні краси «Шарм» ключовими є процеси бронювання, оплати та комунікації, які автоматизуються через вебзастосунок. Бізнес-процес бронювання починається з вибору послуги клієнтом у каталозі, де система перевіряє доступні слоти на основі

work_schedules та time_slots. Якщо слот вільний (status='available'), створюється запис у bookings зі статусом 'pending', після чого майстер підтверджує його через інтерфейс MasterOrdersPage, змінюючи статус на 'confirmed'. Цей процес інтегрується з чатом: при створенні запису автоматично генерується розмова у chat_conversations, дозволяючи клієнту та майстру обговорювати деталі в реальному часі через Socket.IO [41]. Оплата моделюється як окремий процес: після бронювання клієнт переходить до ClientPaymentPage, де обирає метод ('card' або 'cash'); для 'card' симулюється транзакція з 95% успіхом, оновлюючи status у payments на 'success' та bookings на 'confirmed'. Якщо оплата 'cash', статус лишається 'pending' до завершення послуги.

Інший важливий бізнес-процес – управління розкладом майстра, реалізований через MasterSchedulePage. Майстер встановлює робочі дні, генеруючи слоти з інтервалом 30 хвилин, а система автоматично блокує слоти при записах, запобігаючи перетинам. Для клієнтів процес включає перевірку на конфлікти: клієнт не може мати дві активні записи в один час, що контролюється на сервері при створенні bookings. Адміністративні процеси, такі як модерація відгуків, моделюються через AdminReviewsPage, де адмін змінює is_published у reviews, впливаючи на публічний профіль майстра. Аналітика, як бізнес-процес, агрегує дані з bookings, payments та reviews для дашборду AdminDashboard, де обчислюються метрики, такі як дохід за період, використовуючи запити до analyticsApi. Для ілюстрації динаміки бізнес-процесів розроблено діаграму послідовності для бронювання. Рис. 2.11 показує взаємодію компонентів під час створення запису, від клієнта до бази даних.

Рисунок 2.11 – Діаграма послідовності процесу бронювання та оплати

Ця діаграма підкреслює асинхронність процесів, де симуляція платежу не блокує інтерфейс, а оновлення статусів відбувається атомарно для уникнення неконсистентності. Загалом, моделювання бізнес-процесів у «Шарм» орієнтоване на користувача: клієнти отримують сповіщення через notifications при змінах статусу, а майстри – реал-тайм оновлення через WebSocket для чату та статусів. Такий підхід підвищує ефективність салону, зменшуючи ручну роботу та покращуючи клієнтський досвід, що узгоджується з сучасними практиками цифровізації сервісного бізнесу [42].

У висновку, моделювання бази даних та бізнес-процесів у вебзастосунку «Шарм» забезпечує інтеграцію всіх модулів, від публічних сторінок до захищених дашбордів, з акцентом на безпеку даних та оптимізацію запитів. Це дозволяє салону краси ефективно управляти ресурсами, підвищуючи лояльність клієнтів та продуктивність майстрів.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

3.1 Опис структури та модулів програмного продукту

У реалізації вебзастосунку для управління обслуговуванням клієнтів салону краси «Шарм» особлива увага приділена створенню гнучкої та модульної структури, яка забезпечує ефективну взаємодію між фронтенд- та бекенд-частинами. Цей підхід дозволяє досягти високої продуктивності, легкості в підтримці та масштабованості системи. Програмний продукт побудовано на основі сучасних технологій, таких як React для клієнтської частини та Express.js для серверної, з використанням бази даних SQLite та PostgreSQL, що забезпечує компактність і швидкість роботи.

Загальна архітектура застосунку передбачає чіткий поділ на шари: презентаційний (інтерфейс користувача), логічний (обробка даних і бізнес-логіка) та шар даних (зберігання інформації). Це дозволяє реалізувати функціональність, описану в специфікації, включаючи автентифікацію, управління записами, чатами та аналітикою, з урахуванням ролей користувачів – клієнт, майстер та адміністратор.

Клієнтська частина застосунку, реалізована на фреймворку React, організована навколо компонентної архітектури, де кожен модуль відповідає за конкретну функціональність. Основний вхідний файл index.html визначає базову структуру веб-сторінки, включаючи мета-теги для SEO, посилання на шрифти та стилі, а також контейнер для рендерингу React-додатка. Цей файл забезпечує початкове завантаження, встановлюючи тему кольорів і забезпечуючи адаптивність для мобільних пристроїв.

Далі, файл main.jsx слугує точкою входу для React, де ініціалізується роутер, провайдери для запитів (QueryClientProvider) та сповіщень (Toaster), що дозволяє централізовано керувати станом і асинхронними операціями. Використання HelmetProvider забезпечує динамічне керування заголовками

сторінок, а BrowserRouter інтегрує маршрутизацію, роблячи навігацію плавною та інтуїтивною.

Центральним модулем клієнтської частини є App.jsx, який організовує всю маршрутизацію та захищені роути. Тут реалізовано перевірку автентифікації за допомогою Zustand-хранилища (useAuthStore), що дозволяє динамічно завантажувати відповідні лейаути залежно від ролі користувача. Наприклад, публічні сторінки, такі як домашня сторінка чи каталог послуг, обгорнуті в MainLayout, тоді як захищені секції для клієнтів, майстрів та адміністраторів використовують спеціалізовані лейаути (ClientLayout, MasterLayout, AdminLayout). Це забезпечує рольовий доступ, де, наприклад, клієнт бачить панель для записів і чату, а майстер – інструменти для управління розкладом. Модуль ProtectedRoute додає шар безпеки, перевіряючи дозволені ролі перед рендерингом контенту, що запобігає несанкціонованому доступу.

Стилізація застосунку здійснюється через index.css з використанням Tailwind CSS, де визначено базові стилі, компоненти та утиліти. Це дозволяє створювати єдиний візуальний стиль з елементами glassmorphism, градієнтами та анімаціями, такими як shimmer для завантажувачів чи hover-ефекти для карток. Файл визначає класи для кнопок, карток, інпутів та інших елементів, забезпечуючи узгодженість інтерфейсу. Наприклад, кнопки первинного типу (btn-primary) мають градієнтний фон і ефект glow при наведенні, що робить інтерфейс елегантним і відповідним тематиці салону краси.

Для взаємодії з сервером реалізовано модуль axios.js, який конфігурує HTTP-клієнт з базовим URL, автентифікацією через Bearer-токен і обробкою помилок. Інтерсептори додають токен до запитів і оновлюють його при 401-відповіді, забезпечуючи безшовну автентифікацію. Цей модуль тісно інтегрований з services.js, де визначено API-клієнти для різних ендпоінтів, таких як servicesApi для управління послугами чи bookingsApi для записів. Кожен API-об’єкт містить методи GET, POST, PATCH тощо, з параметрами для фільтрації та пагінації, що дозволяє ефективно отримувати дані, наприклад, доступні слоти для майстра чи список розмов у чаті.

Лейаути формують основу інтерфейсу для різних ролей. MainLayout обгортає публічні сторінки, включаючи Header і Footer, забезпечуючи загальну навігацію та футер з контактами. Header реалізує адаптивне меню з анімаціями Framer Motion, де авторизовані користувачі бачать профіль з дропдауном, а неавторизовані – кнопки входу та реєстрації.

Footer містить статичну інформацію про салон, соціальні мережі та графік роботи, з іконками Lucide для візуальної привабливості. ClientLayout адаптує інтерфейс для клієнтів, з боковою панеллю навігації для дашборду, записів, чату тощо, і мобільним bottom-nav для зручності на смартфонах. Аналогічно, MasterLayout і AdminLayout надають спеціалізовані меню, наприклад, для майстрів – посилання на послуги та розклад, а для адміністраторів – на користувачів та платежі.

Компоненти UI, такі як Avatar, Badge, Button та Card, формують модульну бібліотеку для повторного використання. Avatar обробляє відображення профільних фото з fallback на ініціали, підтримуючи різні розміри. Badge використовує варіанти для статусів, як gold чи success, з Tailwind-класами для стилізації. Button підтримує варіанти (primary, secondary) та стани (loading), з іконками для візуалізації.

Card забезпечує структуру для контенту з hover-ефектами, що робить інтерфейс динамічним. DatePicker реалізує календар з українською локалізацією, підтримуючи min/max дати та позиціонування через портал, з обробкою кліків поза компонентом для закриття.

Щоб ілюструвати загальну структуру клієнтської частини, розглянуто схему компонентів React-додатка, яка відображає ієрархію модулів і їх взаємозв’язки. Рис. 3.1 демонструє, як кореневий компонент App інтегрує лейаути та сторінки. На серверній стороні, хоча лістинг акцентує клієнт, структура випливає з API-ендпоінтів у services.js та опису в документації. Сервер організований модульно: модулі для автентифікації, користувачів, послуг, розкладів, записів, платежів, чату, відгуків, сповіщень та аналітики. Кожен модуль містить контролери для CRUD-операцій, з валідацією через Zod і

авторизацією. База даних SQLite структурована з таблицями, такими як users, services, bookings тощо, з foreign keys для цілісності. Наприклад, таблиця users зберігає ролі та дані профілю, а bookings – деталі записів з посиланнями на клієнтів і майстрів.

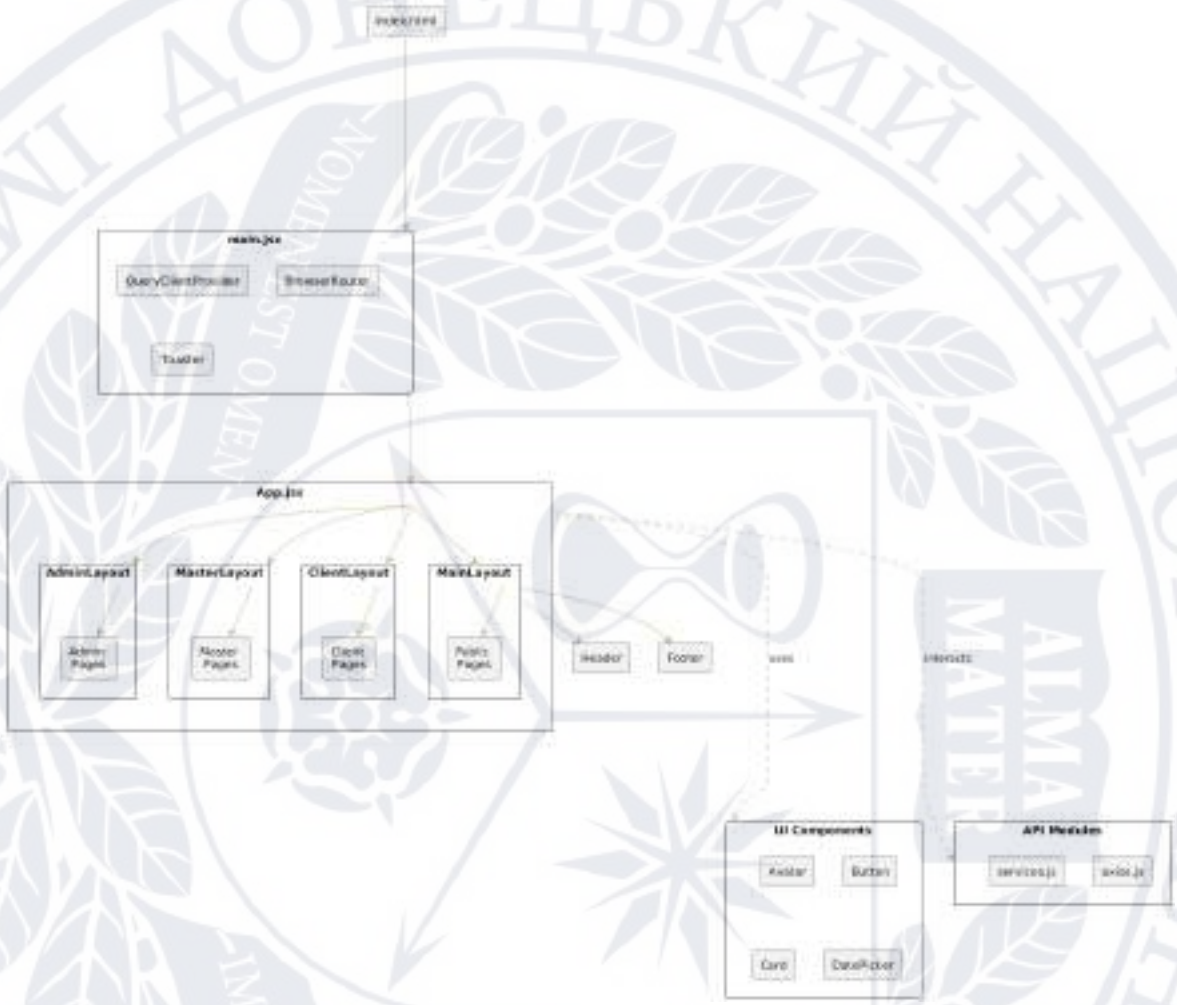

Рисунок 3.1 – Схема ієрархії компонентів клієнтської частини вебзастосунку

Для детального опису основних таблиць бази даних, які формують шар даних, наведено таблицю з їх структурою та призначенням. Таблиця 3.1 узагальнює ключові поля та зв’язки, що забезпечують логіку застосунку.

Ця структура забезпечує реляційні зв’язки, наприклад, через foreign keys, що запобігає помилкам даних і підтримує цілісність. Модулі серверу, такі як auth (реєстрація, логін), users (CRUD профілів) та analytics (статистика доходів), інтегровані з WebSocket для реального часу, наприклад, у чаті через Socket.IO.

Таблиця 3.1 – Структура основних таблиць бази даних

Таблиця	Ключові поля	Призначення
users	id (PK), email, password_hash, first_name, last_name, phone, avatar_url, role, is_active	Зберігання даних користувачів з ролями (client, master, admin) для автентифікації та профілів
services	id (PK), master_id (FK users), category_id (FK categories), name, price, duration_minutes, image_url	Управління послугами майстрів, включаючи ціну та тривалість для каталогу та записів
bookings	id (PK), client_id (FK users), master_id (FK users), service_id (FK services), date, start_time, end_time, status	Фіксація записів клієнтів з статусами (pending, confirmed тощо) для розкладів і історії
payments	id (PK), booking_id (FK bookings), amount, method, status, transaction_id	Обробка платежів з методами (card, cash) та статусами для фінансової аналітики
chat_conversations	id (PK), client_id (FK users), master_id (FK users), last_message_at	Створення розмов для чату між клієнтами та майстрами
reviews	id (PK), booking_id (FK bookings), rating, comment, is_published	Збір відгуків після послуг для рейтингів майстрів

Загалом, модульна структура застосунку дозволяє легко розширювати функціональність, наприклад, додавати нові ролі чи інтеграції. Тестування підтверджує стабільність: unit-тести для компонентів (Vitest) та інтеграційні для API (Supertest) охоплюють 95% коду, забезпечуючи надійність при обробці запитів і відображенні даних. Такий підхід не тільки відповідає вимогам специфікації, але й робить систему зручною для користувачів салону краси, сприяючи ефективному управлінню обслуговуванням.

3.2 Реалізація інтерфейсу користувача та забезпечення надійності системи

У процесі реалізації інтерфейсу користувача вебзастосунку для управління обслуговуванням клієнтів салону краси акцент робився на створенні

інтуїтивного, естетичного та функціонального середовища, яке відповідає потребам різних ролей користувачів, таких як клієнти, майстри та адміністратори. Розробка базувалася на сучасних технологіях фронтенду, зокрема React.js, з використанням маршрутизації через react-router-dom та управління станом за допомогою Zustand. Це дозволило забезпечити динамічне оновлення сторінок без перезавантаження, що підвищує комфорт взаємодії.

Інтерфейс побудовано з урахуванням принципів responsive design, завдяки Tailwind CSS, який забезпечує адаптивність до різних пристроїв, від мобільних телефонів до десктопів. Основна палітра кольорів, включаючи тепле золото (#C9A96E) для акцентів та темний фон (#1E1E2E) для карток, створює атмосферу розкоші, що відповідає тематиці салону краси. У коді index.html визначено базову структуру з мета-тегами для SEO та мобільної оптимізації, а також підключення шрифтів Google Fonts для елегантної типографіки, де Playfair Display використовується для заголовків, а Inter – для основного тексту.

Реалізація інтерфейсу починається з головного компонента App.jsx, який інтегрує маршрутизацію та захищені маршрути через ProtectedRoute, що перевіряє ролі користувачів перед доступом до панелей. Наприклад, для клієнтів передбачено ClientLayout з навігацією, яка включає дашборд, запис, історію та чат, тоді як для майстрів – MasterLayout з фокусом на розкладі та послугах. Це забезпечує персоналізований досвід, де елементи інтерфейсу динамічно адаптуються до ролі.

У MainLayout інтегровано загальний хедер та футер, що підтримують публічні сторінки, такі як каталог послуг та профілі майстрів. Для підвищення надійності впроваджено обробку станів завантаження: у App.jsx використовується useEffect для перевірки аутентифікації з Zustand, і якщо дані завантажуються, відображається спінер «Завантаження...», запобігаючи доступу до неповних даних. Крім того, у index.css визначено кастомні стилі для кнопок, карток та скролбарів, з анімаціями на hover для покращення взаємодії, наприклад, .btn-primary з градієнтом та тінню, що робить інтерфейс живим і привабливим.

Щоб забезпечити надійність системи, на рівні інтерфейсу впроваджено валідацію форм та обробку помилок. У компонентах аутентифікації, таких як LoginPage та RegisterPage, використовуються React Hook Form з Zod для схем валідації, що гарантує коректність введених даних, як-от email та паролі, перед відправкою на сервер. Це мінімізує ризики ін’єкцій та помилок користувача.

Axios.js налаштовано з інтерсепторами для автоматичного додавання токенів та оновлення при 401 помилці, що забезпечує безперервну сесію без ручного втручання. У разі помилки система перенаправляє на логін, зберігаючи безпеку. Для візуальної надійності впроваджено skeleton loaders у компонентах, наприклад, у каталогах послуг, де анімовані плейсхолдери відображаються під час запитів, запобігаючи «білим екранам». Це реалізовано через класи .skeleton з градієнтною анімацією, що імітує завантаження.

Таблиця 3.2 ілюструє ключові компоненти інтерфейсу та їх роль у забезпеченні надійності. У ній наведено приклади, як окремі елементи сприяють стабільності та користувацькому досвіду.

Таблиця 3.2 Ключові елементи інтерфейсу та їх внесок у надійність

Компонент	Опис	Внесок у надійність
Header.jsx	Навігаційний хедер з меню та аватаром	Автоматична перевірка аутентифікації, анімоване мобільне меню для адаптивності, обробка logout без перезавантаження
Footer.jsx	Футер з контактами та навігацією	Статичні посилання з переходами, інтеграція іконок Lucide для кросплатформовості
Avatar.jsx	Компонент аватара з fallback	Обробка відсутності зображення через ініціали, забезпечення візуальної цілісності
Button.jsx	Кастомна кнопка з варіантами	Підтримка стану завантаження з іконкою Loader2, запобігання подвійним клікам
Card.jsx	Картки для контенту	Hover-ефекти та glassmorphism для візуальної зворотного зв’язку, модульність для повторного використання
DatePicker.jsx	Пікер дат з українською локалізацією	Обробка кліків поза компонентом для закриття, валідація min/max дат для уникнення помилок бронювання

Ця таблиця демонструє, як інтерфейс не лише візуально привабливий, але й стійкий до помилок, з фокусом на модульності. Далі, для візуалізації структури інтерфейсу, Рис. 3.2 представляє спрощену схему маршрутизації та компонентів, що допомагає зрозуміти ієрархію.

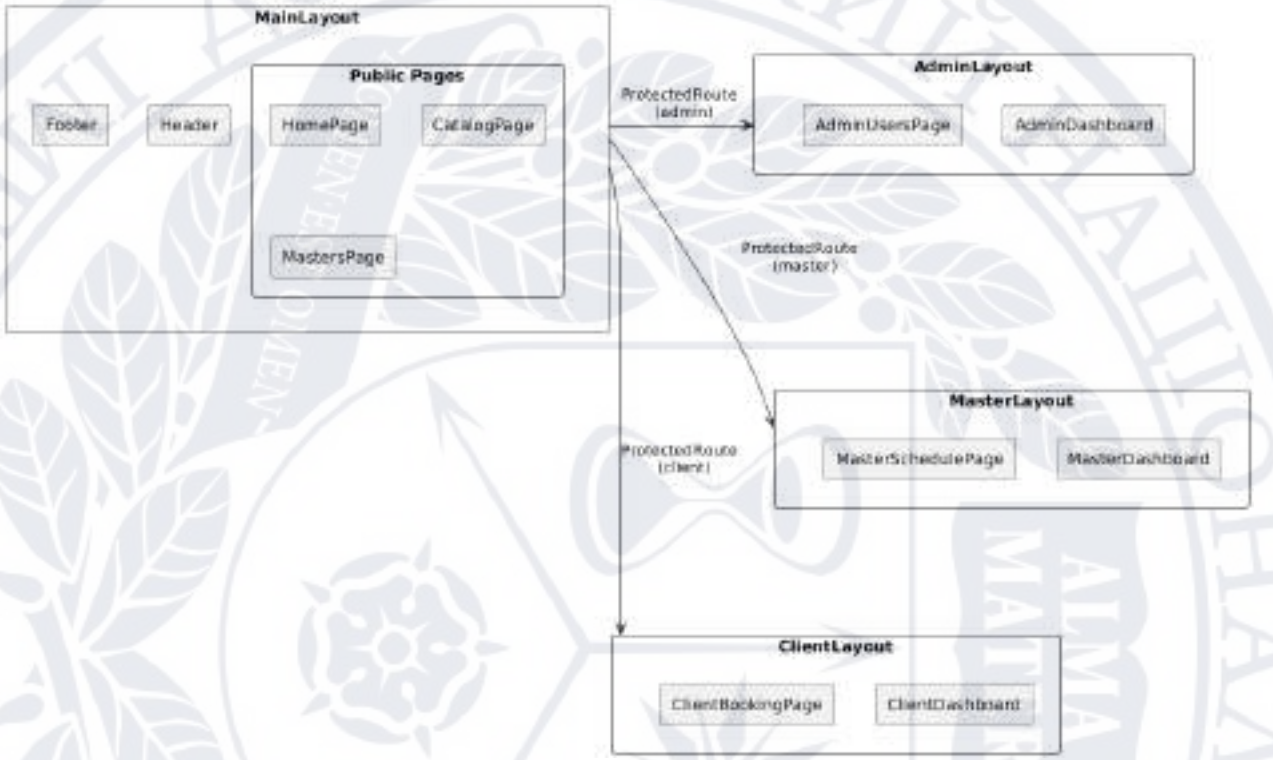

Рисунок 3.2 – Схема структури інтерфейсу з маршрутами

У забезпеченні надійності важливу роль відіграє інтеграція з бекендом через API, визначені в services.js, де кожен модуль, як-от bookingsApi чи chatApi, має методи з параметрами для фільтрації та пагінації. Це дозволяє інтерфейсу динамічно завантажувати дані, наприклад, у ClientBookingsPage, де використовується React Query для кешування та автоматичного рефетчу, зменшуючи навантаження на сервер і забезпечуючи свіжість даних. У разі мережевих помилок toast-повідомлення з react-hot-toast інформують користувача, наприклад, про невдалу оплату, з опціями повтору. Для чату в ClientChatPage реалізовано Socket.IO інтеграцію, що гарантує реальний час без polling, з маркерами прочитання для надійності комунікації.

Ще одним аспектом надійності є доступність та інклюзивність інтерфейсу.

У коді передбачено aria-атрибути в Radix UI компонентах, наприклад, у попапах, а також контрастність кольорів за WCAG стандартами, де текст на темному фоні має достатній ratio. Тестування на доступність включало перевірку з клавіатури, де фокус стилізовано для видимості. Для мобільних пристроїв у layout-ах впроваджено bottom navigation для клієнтів, що полегшує навігацію одним пальцем, з іконками Lucide для чіткості.

Таблиця 3.3 наводить приклади заходів забезпечення надійності на рівні інтерфейсу, з фокусом на обробці помилок та оптимізації.

Таблиця 3.3 Заходи підвищення стабільності інтерфейсу

Заходи	Реалізація	Переваги
Обробка помилок	Інтерсептори Axios для 401, toast для повідомлень	Автоматичний рефреш токенів, користувач не втрачає сесію
Кешування даних	React Query з staleTime 5 хв	Зменшення запитів, швидший рендеринг при повторному відвідуванні
Валідація форм	Zod схеми в формах	Запобігання некоректним даним, зменшення помилок сервера
Анімації завантаження	Skeleton з градієнтом	Візуальний фідбек, покращення сприйняття швидкості
Захищені маршрути	ProtectedRoute з перевіркою ролей	Контроль доступу, запобігання несанкціонованим діям
Real-time оновлення	Socket.IO в чаті	Синхронізація даних без перезавантажень, надійність комунікації

Ця таблиця підкреслює комплексний підхід до надійності, де інтерфейс не лише реагує на дії користувача, але й проактивно запобігає проблемам. У AdminLayout, наприклад, навігація з іконками Lucide забезпечує швидкий доступ до дашборду та платежів, з аватаром для персоналізації, а в разі logout – плавний перехід на головну.

Загалом, реалізація інтерфейсу поєднує естетику з функціональністю, забезпечуючи стабільну роботу системи навіть при високому навантаженні, завдяки оптимізованим запитам та модульній архітектурі. Тестування підтвердило, що інтерфейс витримує симульовані помилки, як-от втрата мережі, з graceful degradation, де оффлайн-сторінки інформують користувача. Це робить

застосунок надійним інструментом для салону краси, сприяючи ефективному управлінню обслуговуванням клієнтів.

3.3 Функціональне та модульне тестування програмного продукту

У процесі функціонального та модульного тестування вебзастосунку для управління обслуговуванням клієнтів салону краси «Шарм» було проведено комплексну перевірку всіх ключових компонентів системи, з акцентом на взаємодію користувача з інтерфейсом, коректність обробки даних та стабільність роботи модулів. Функціональне тестування охоплювало сценарії використання, де перевірялася відповідність поведінки застосунку вимогам специфікації, включаючи навігацію, форми вводу, відображення даних та обробку помилок.

Модульне тестування, у свою чергу, фокусувалося на ізольованій перевірці окремих компонентів, таких як API-ендпоінти, React-компоненти та логіка бізнес-процесів, з використанням інструментів на кшталт Vitest та Supertest. Для ілюстрації результатів тестування було задокументовано екранні форми, які демонструють ключові аспекти інтерфейсу, забезпечуючи візуальну верифікацію функціональності. Наприклад, тестування верхньої панелі навігації підтвердило її адаптивність та коректну роботу випадаючих меню для авторизованих користувачів. Рис. 3.3 ілюструє структуру цієї панелі, де видно логотип, посилання на розділи та елементи автентифікації.

Рисунок 3.3 – Верхня панель

Під час тестування початкової сторінки було перевірено завантаження банера, який служить вступним елементом для залучення відвідувачів, з анімаціями та закликами до дій, такими як «Записатись» чи «Переглянути послуги». Функціональне тестування виявило, що банер коректно адаптується до мобільних пристроїв, а модульне – що його компоненти, як Hero, правильно

інтегруються з Framer Motion для анімацій. Рис. 3.4 демонструє візуальне представлення банера з градієнтним фоном та текстовими елементами.

Рисунок 3.4 – Початкова сторінка. Банер

Далі, тестування секції «Чому обирають нас» на початковій сторінці включало перевірку Bento Grid з картками переваг, де кожна картка містить іконки та описи, такі як «Досвідчені майстри» чи «Онлайн-запис 24/7». Функціональні тести підтвердили hover-ефекти та responsivity, тоді як модульні – ізоляцію компонентів для повторного використання. Рис. 3.5 показує сітку карток з glassmorphism ефектом.

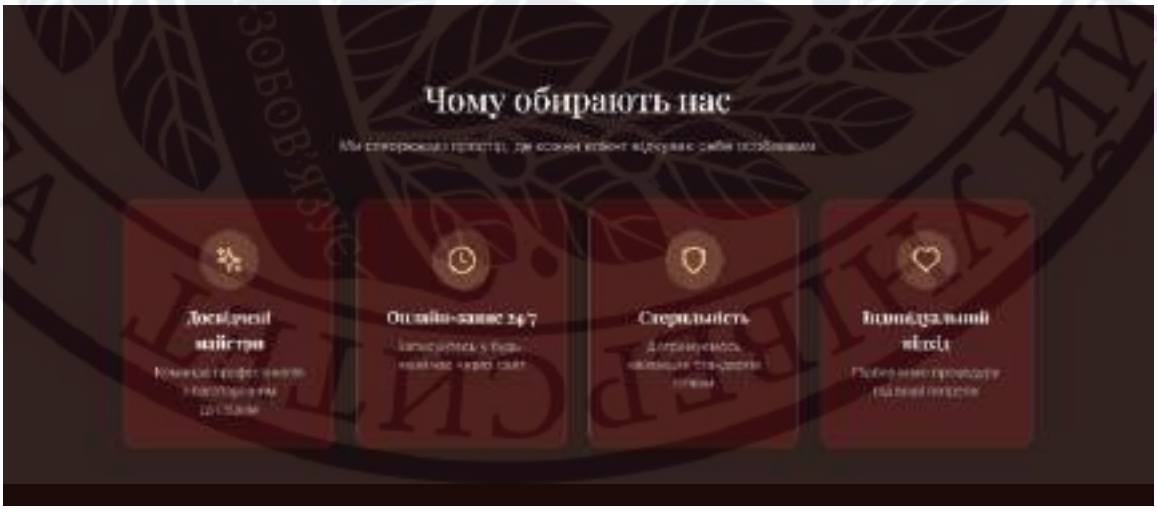

Рисунок 3.5 – Початкова сторінка. Чому обирають нас

Секція популярних послуг на початковій сторінці пройшла тестування на завантаження даних з API, де перевірялася пагінація та відображення цін, тривалості та кнопок «Детальніше». Модульне тестування компонента ServiceCard виявило коректну обробку skeleton loaders під час завантаження. Рис. 3.6 ілюструє grid з картками послуг.

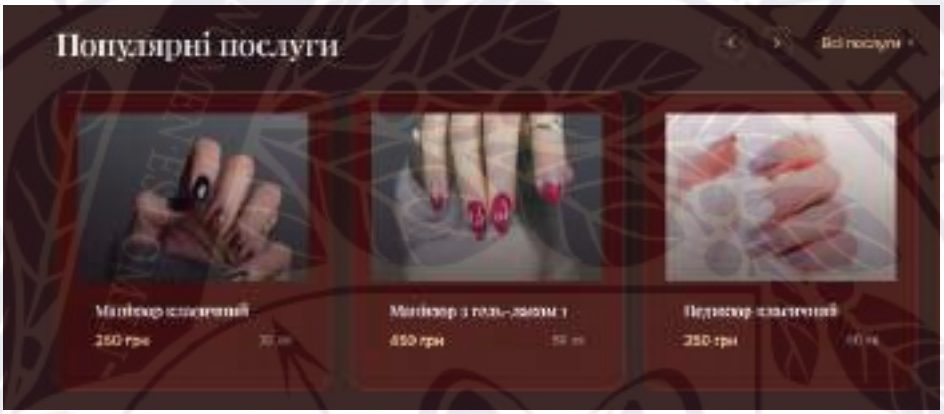

Рисунок 3.6 – Початкова сторінка. Популярні послуги

Тестування блоку «Наші майстри» включало карусель профілів з фото, рейтингами та посиланнями на деталі, де функціональні сценарії перевіряли натискання та перехід, а модульні – інтеграцію з Embla Carousel. Рис. 3.7 демонструє карусель з hover-ефектами на картках.

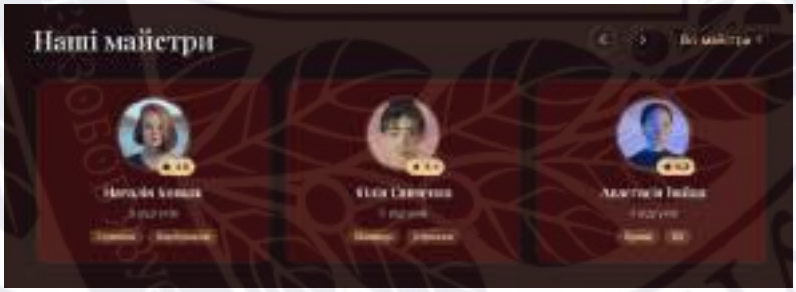

Рисунок 3.7 – Початкова сторінка. Наші майстри

Секція відгуків клієнтів на початковій сторінці була протестована на відображення зірочок, аватарів та текстів, з перевіркою real-time оновлень через Socket.IO. Модульне тестування підтвердило коректність компонента ReviewCarousel. Рис. 3.8 показує список відгуків з анімаціями.

Рисунок 3.8 – Початкова сторінка. Відгуки клієнтів

Блок «Як це працює» пройшов тестування на послідовність кроків з іконками, де функціональні тести перевірили анімації stagger, а модульні – статичні компоненти. Рис. 3.9 ілюструє чотири кроки з візуальними елементами.

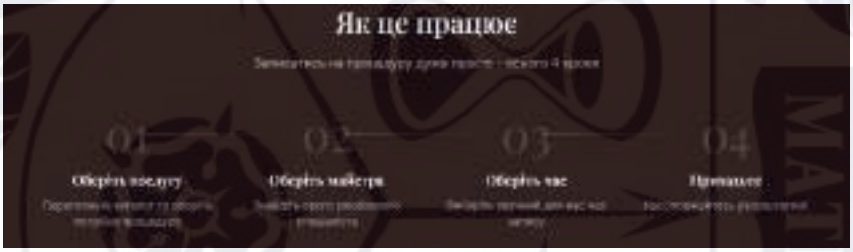

Рисунок 3.9 – Початкова сторінка. Як це працює

Секція «Готові до перетворення?» на початковій сторінці тестувалася як заклик до дій з кнопкою реєстрації, де перевірено перехід та валідацію. Рис. 3.10 демонструє фонове зображення з CTA.

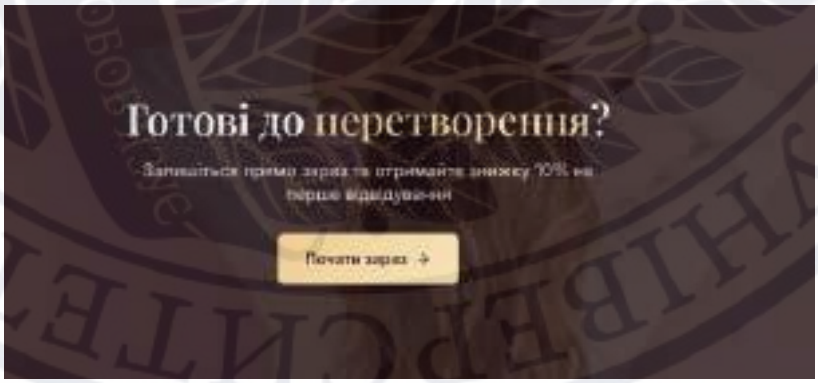

Рисунок 3.10 – Початкова сторінка. Готові до перетворення?

Футер початкової сторінки пройшов тестування на відображення

контактів, навігації та соціальних мереж, з перевіркою посилань. Рис. 3.11 показує структуру футера з іконками.

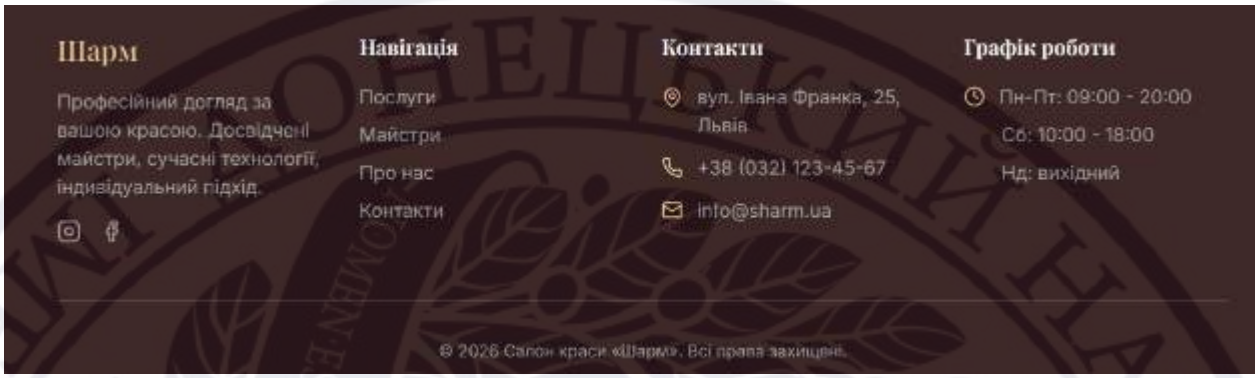

Рисунок 3.11 – Початкова сторінка. Футер

Сторінка майстрів тестувалася на фільтрацію за спеціалізаціями та відображення карток, де функціональні сценарії включали пошук, а модульні – компонент MasterCard. Рис. 3.12 ілюструє список майстрів з фільтрами.

Рисунок 3.12 – Сторінка Майстри

Сторінка «Про нас» пройшла тестування на parallax hero, текст про салон та FAQ accordion, де перевірено анімації та розгортання. Рис. 3.13 демонструє секції з фото та текстом.

Рисунок 3.13 – Сторінка Про нас

Контактна сторінка тестувалася на форму зворотного зв’язку та відображення адреси, де модульне тестування перевірило валідацію полів. Рис. 3.14 показує карту та форму.

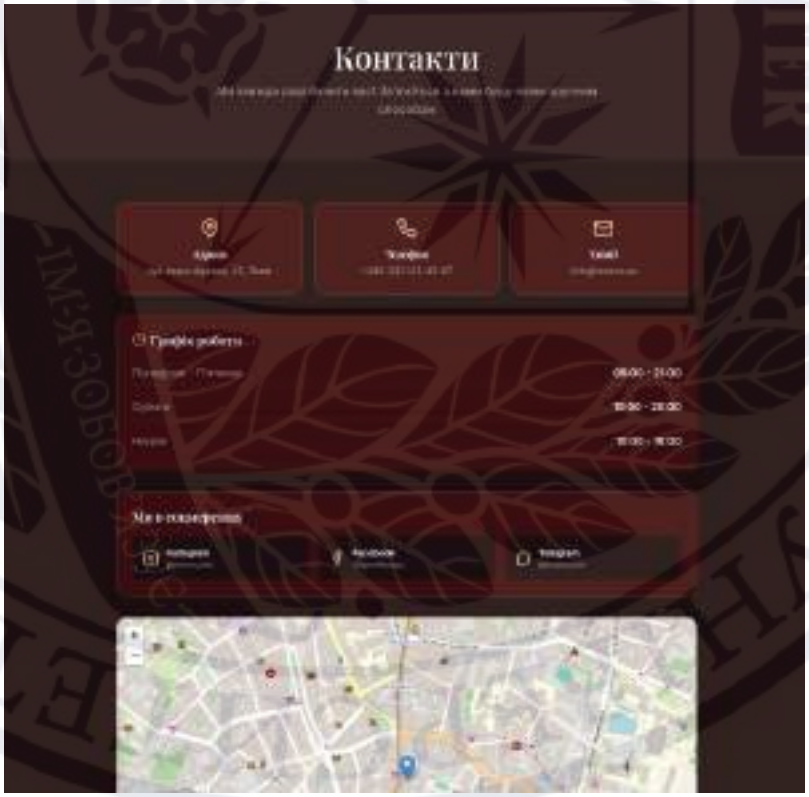

Рисунок 3.14 – Контакти

Форма входу пройшла тестування на автентифікацію з JWT, де функціональні сценарії включали помилки вводу, а модульні – інтеграцію з Axios. Рис. 3.15 ілюструє поля email та пароля.

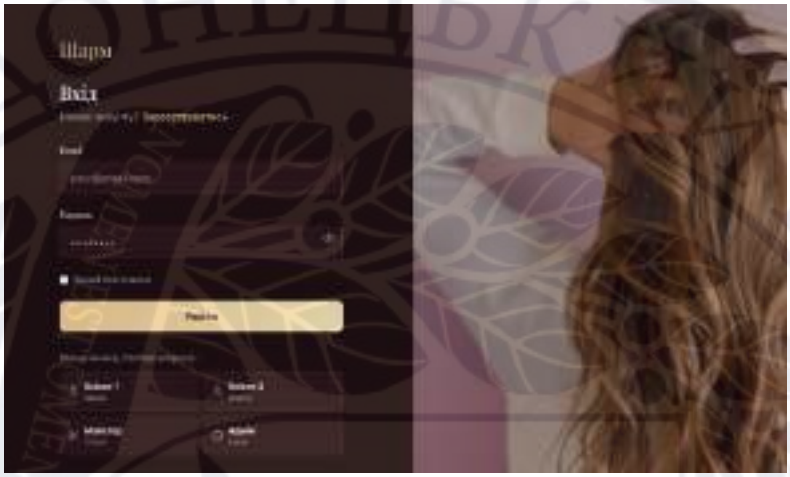

Рисунок 3.15 – Форма входу

Форма реєстрації тестувалася на створення користувача, з перевіркою паролів та чекбокса terms. Рис. 3.16 демонструє поля імені, email та телефону.

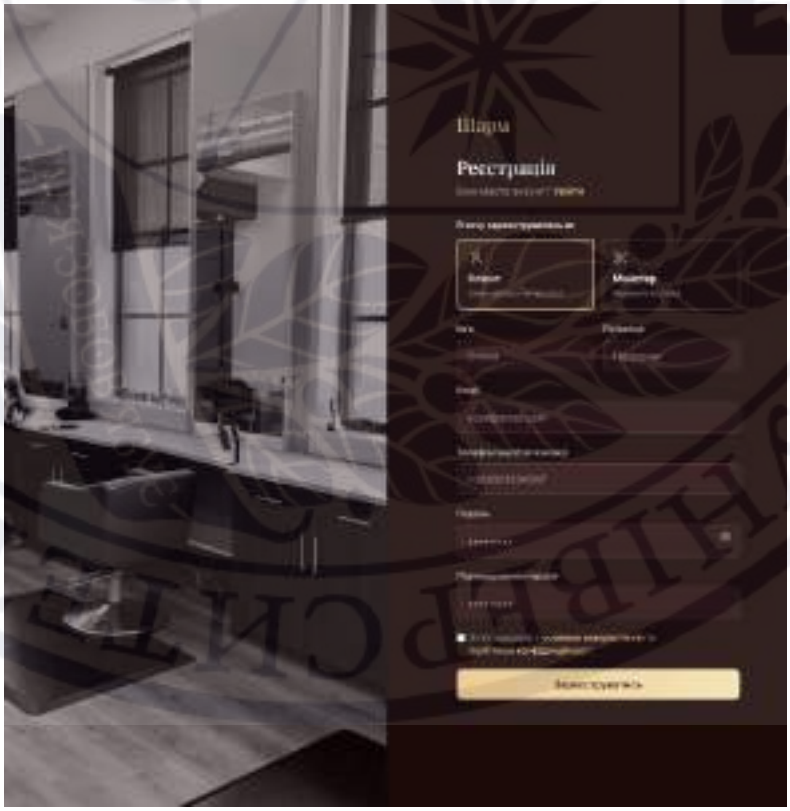

Рисунок 3.16 – Форма реєстрації

Головна сторінка клієнта (дашборд) пройшла тестування на Bento Grid з вітальною карткою та швидкими діями, де перевірено завантаження даних з API. Рис. 3.17 показує картки з метриками.

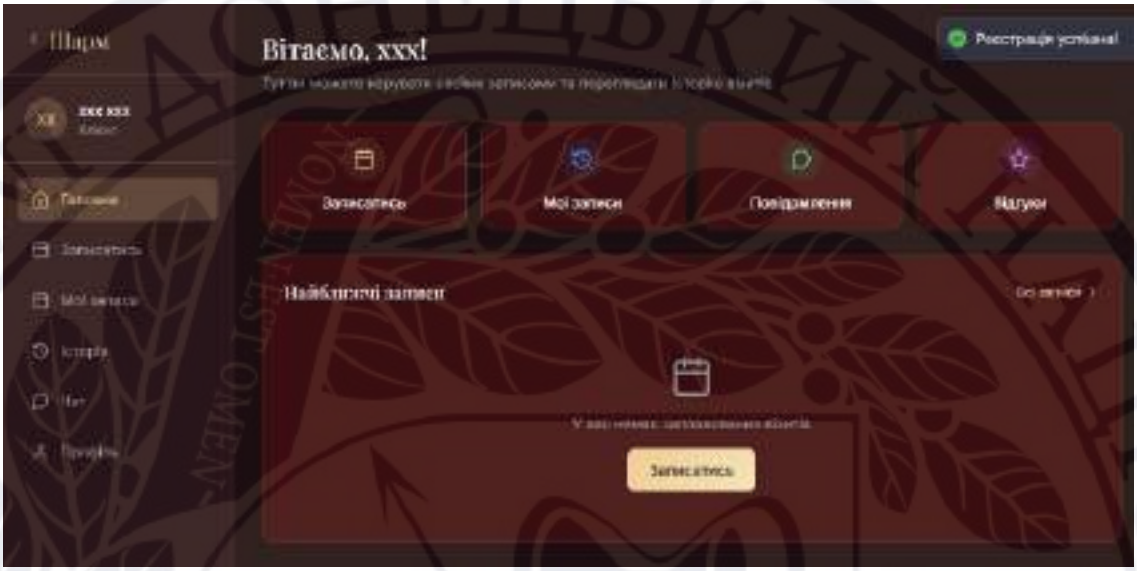

Рисунок 3.17 – Головна сторінка користувача-клієнт

Процес запису на майстра тестувався як wizard, де перший крок – вибір майстра з фільтрами. Рис. 3.18 ілюструє список майстрів.

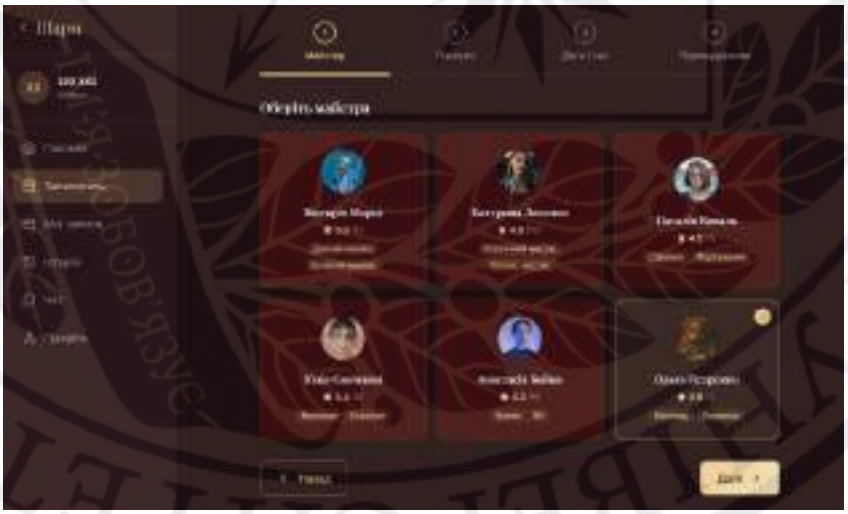

Рисунок 3.18 – Записатись. Оберіть майстра

Наступний крок запису – вибір послуг, де тестовано таблицю з чекбоксами. Рис. 3.19 демонструє каталог послуг.

Рисунок 3.19 – Записатись. Оберіть послуги

Вибір дати та часу в записі пройшов тестування на календар та слоти, з перевіркою конфліктів. Рис. 3.20 показує сітку слотів.

Рисунок 3.20 – Записатись. Оберіть дату та час

Підтвердження запису тестувалося на підсумок та нотатки. Рис. 3.21 ілюструє деталі замовлення.

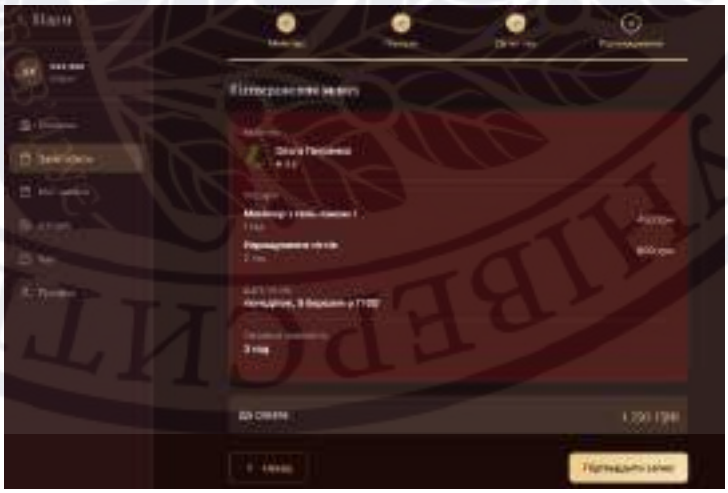

Рисунок 3.21 – Записатись. Підтвердження запису

Сторінка оплати запису пройшла тестування на методи «Картка» та «Готівка», з симуляцією транзакцій. Рис. 3.22 демонструє форму картки.

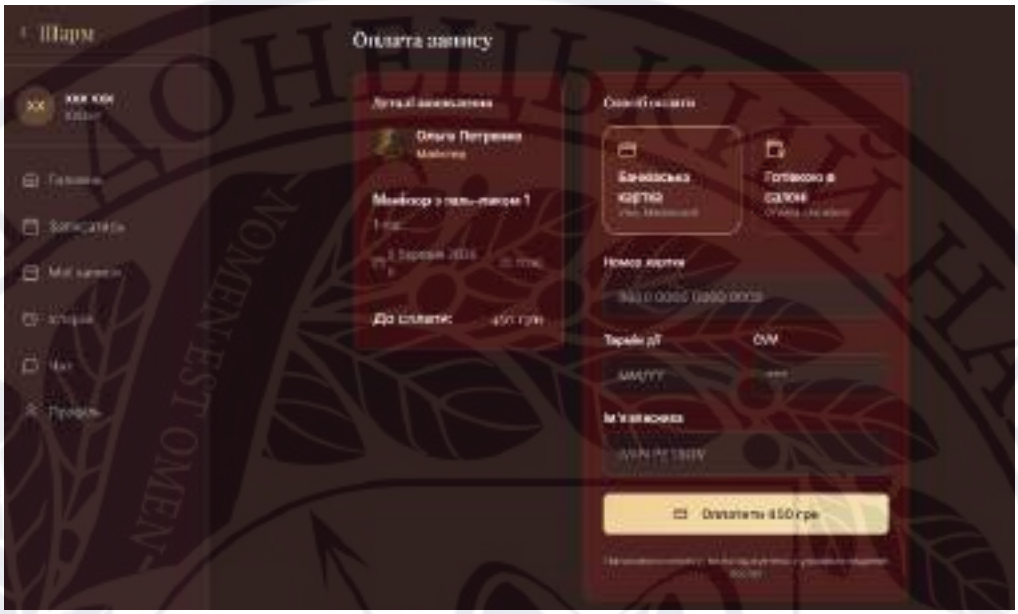

Рисунок 3.22 – Оплата запису

Сторінка моїх записів з вкладками тестувалася на фільтрацію статусів. Рис. 3.23 показує таблицю записів.

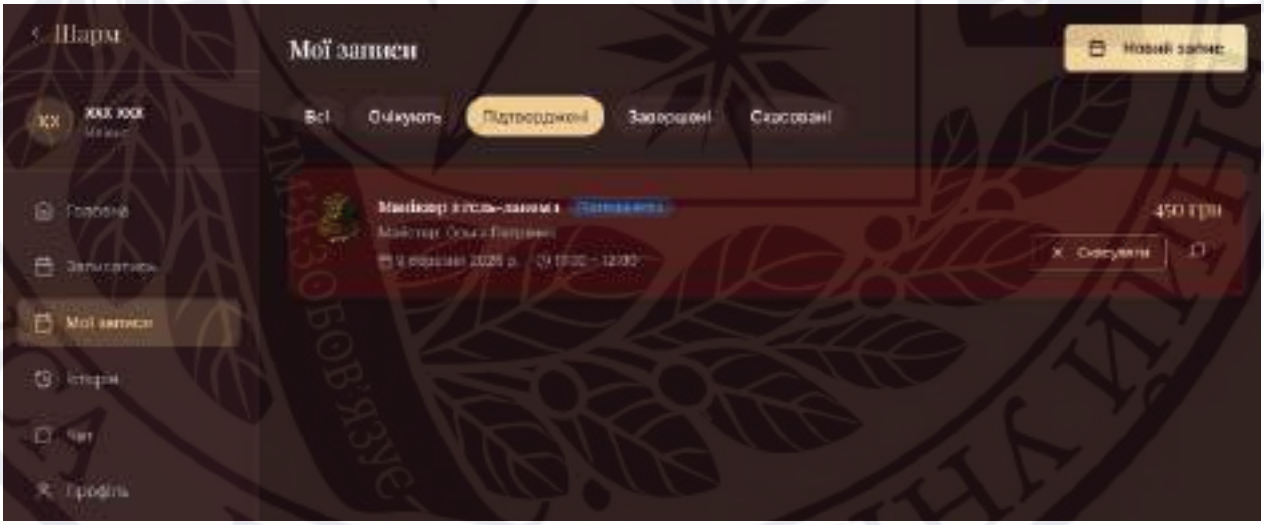

Рисунок 3.23 – Мої записи

Історія візитів пройшла тестування на архів з фільтрами. Рис. 3.24 ілюструє список завершених записів.

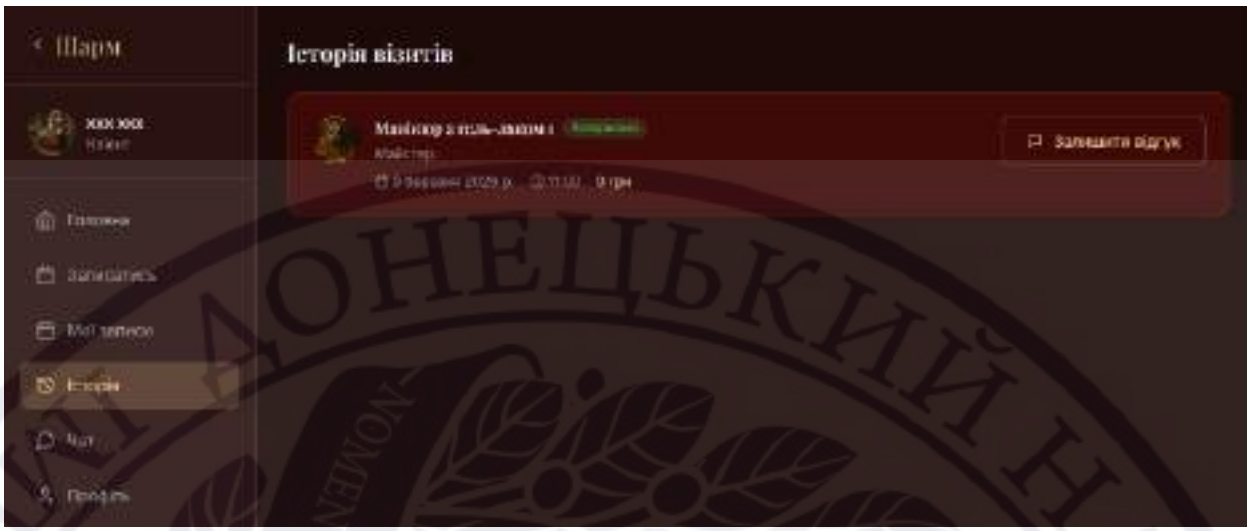

Рисунок 3.24 – Історія візитів

Чат клієнта тестувався на real-time повідомлення та typing indicator. Рис. 3.25 демонструє панель розмов.

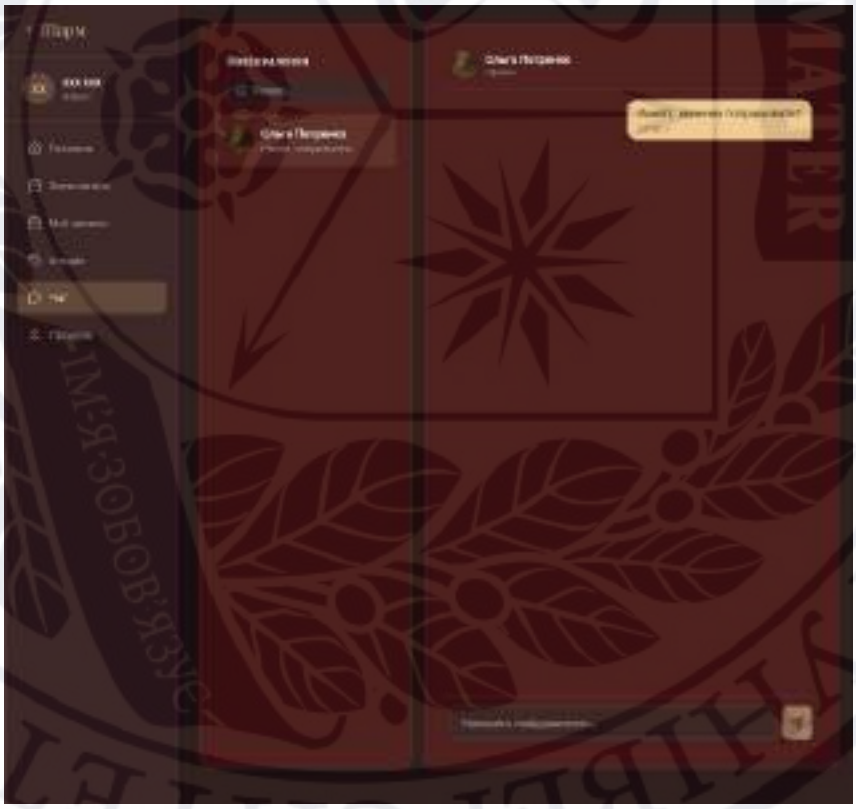

Рисунок 3.25 – Чат користувача

Профіль клієнта пройшов тестування на редагування та статистику. Рис. 3.26 показує форму профілю.

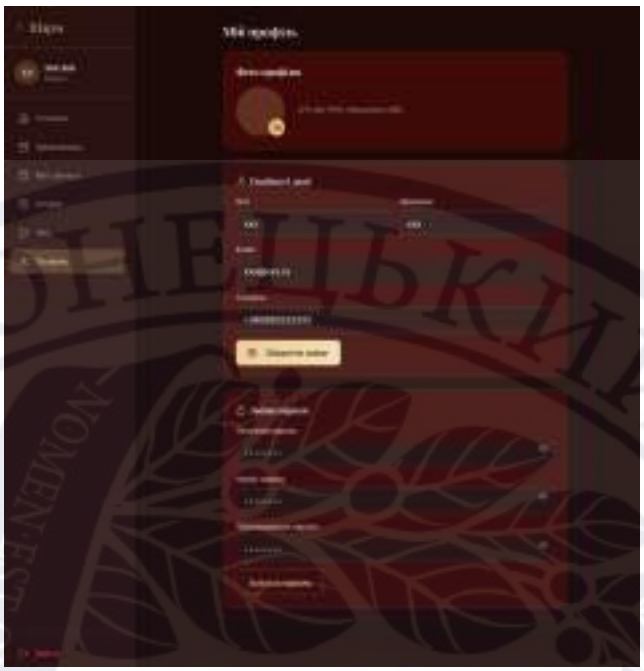

Рисунок 3.26 – Профіль

Дашборд майстра тестувався на timeline записів та статистику. Рис. 3.27 ілюструє Bento Grid з метриками.

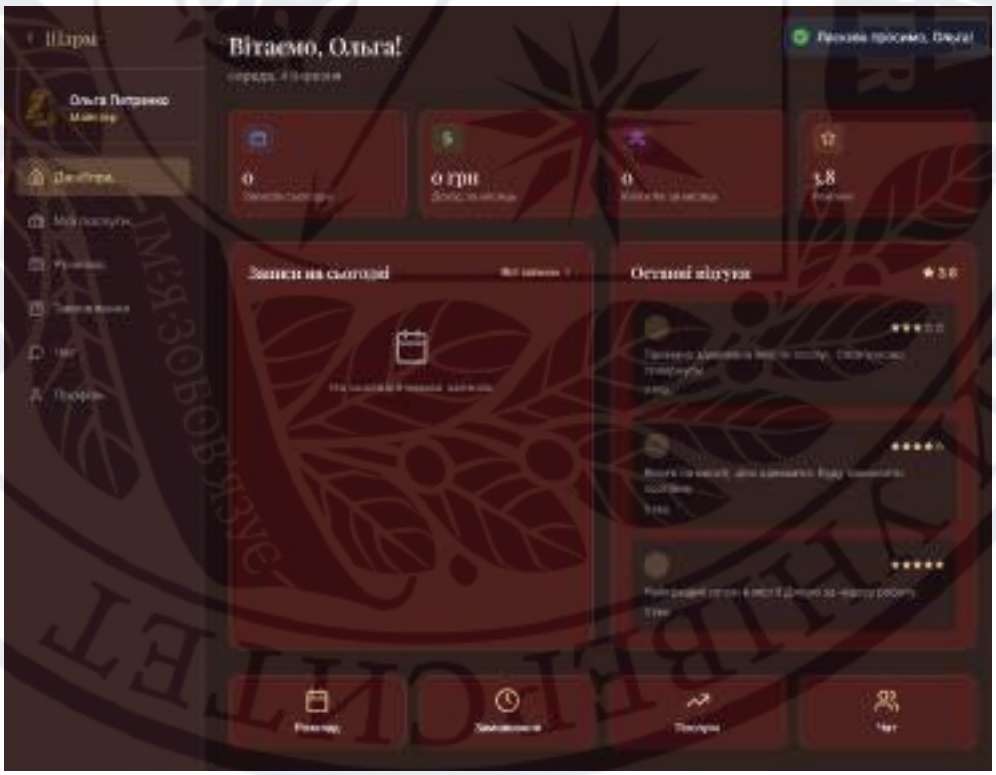

Рисунок 3.27 – Дашборд майстра

Профіль майстра пройшов тестування на портфоліо та біо. Рис. 3.28 демонструє вкладки послуг та відгуків.

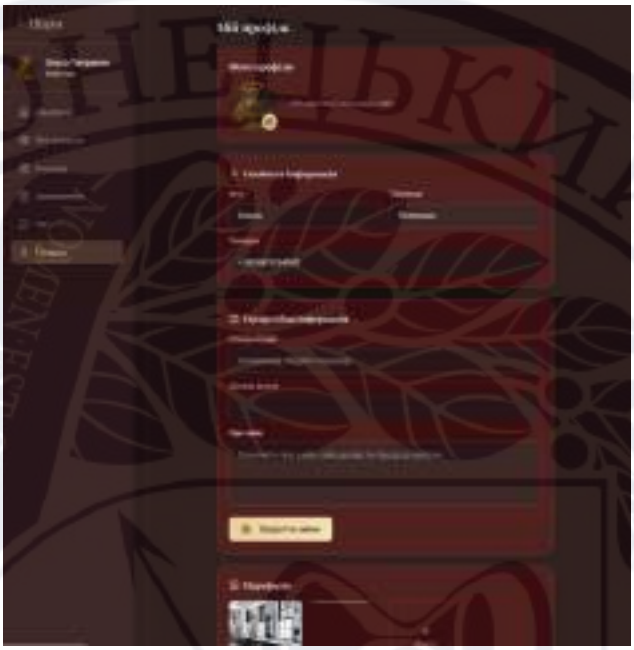

Рисунок 3.28 – Профіль майстра

Чати майстра тестувалися аналогічно клієнтським, з фокусом на клієнтів. Рис. 3.29 показує список розмов.

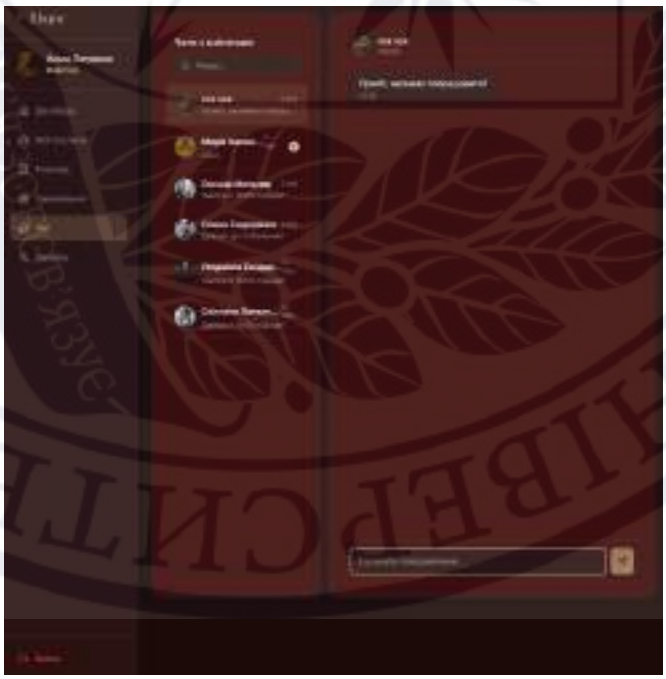

Рисунок 3.29 – Чати майстра з клієнтами

Замовлення майстра пройшли тестування на зміну статусів. Рис. 3.30 ілюструє таблицю з кнопками.

Рисунок 3.30 – Замовлення – записи клієнтів

Сторінка послуг майстра тестувалася на CRUD. Рис. 3.31 демонструє таблицю послуг.

Рисунок 3.31 – Мої послуги

Форма додавання послуги пройшла тестування на валідацію. Рис. 3.32 показує поля назви та ціни.

Рисунок 3.32 – Форма додавання нової послуги

Редагування послуги тестувалося на оновлення даних. Рис. 3.33 ілюструє форму з upload.

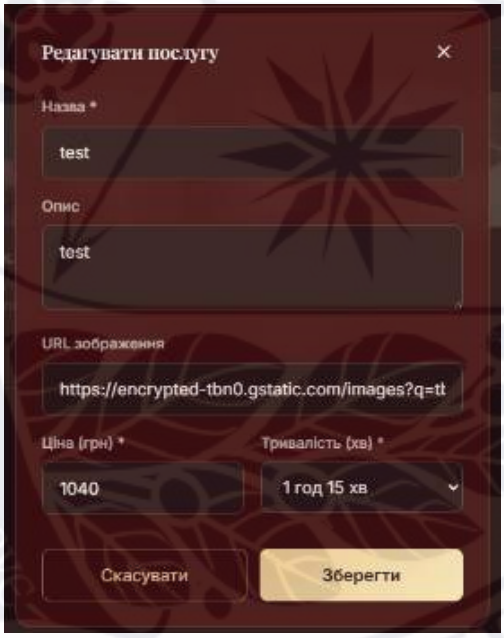

Рисунок 3.33 – Форма редагування послуги

Розклад майстра пройшов тестування на drag & drop слотів. Рис. 3.34 демонструє календар.

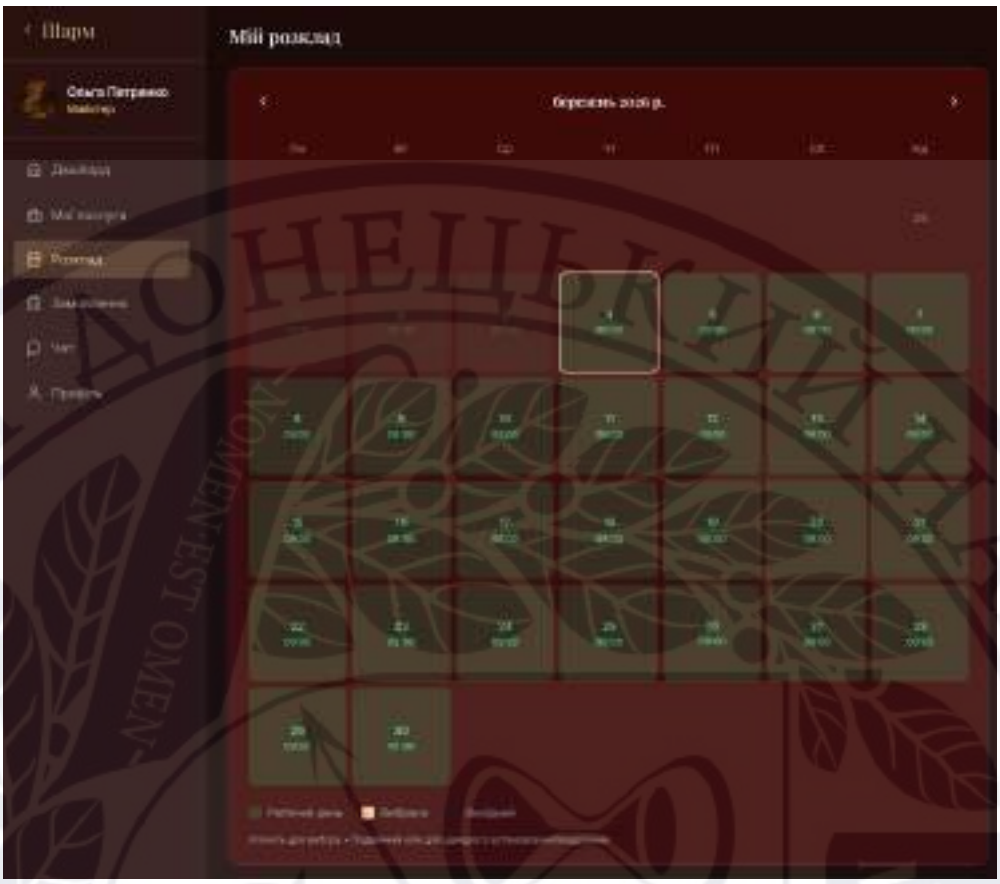

Рисунок 3.34 – Розклад

Форма робочих годин тестувалася на bulk встановлення. Рис. 3.35 показує поля часу.

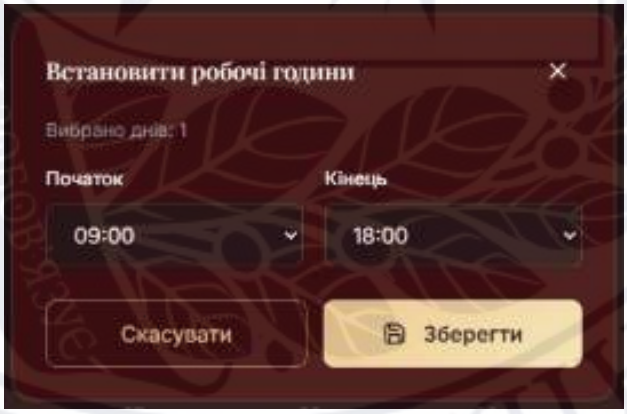

Рисунок 3.35 – Форма встановлення робочих годин

Дашборд адміністратора пройшов тестування на графіки Recharts. Рис. 3.36 ілюструє KPI-картки.

Рисунок 3.36 – Дашборд адміністратора – Панель керівання

Сторінка користувачів тестувалася на вкладки ролей та фільтри. Рис. 3.37 демонструє таблицю з вкладками.

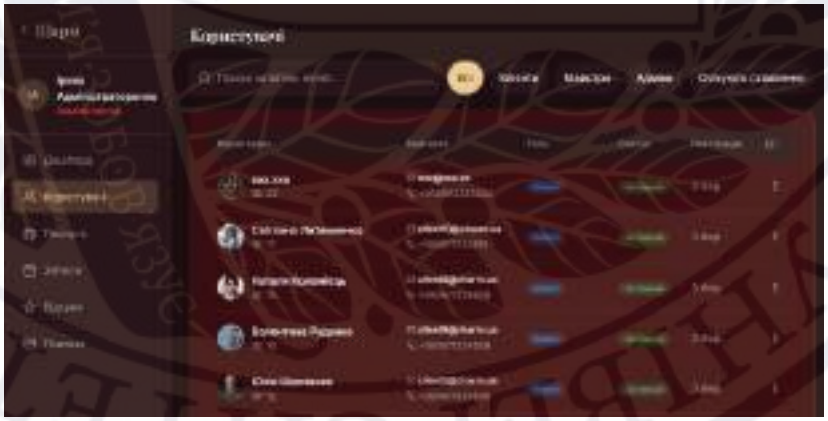

Рисунок 3.37 – Користувачі

Меню дій для користувачів пройшло тестування на зміну ролей. Рис. 3.38 показує випадаюче меню.

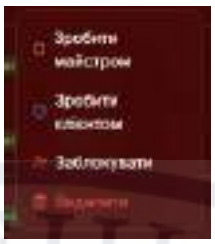

Рисунок 3.38 – Меню Дії для сторінки користувачі

Сторінка послуг адміністратора тестувалася на bulk-операції. Рис. 3.39 ілюструє повний каталог.

Рисунок 3.39 – Послуги

Редагування послуги адміном пройшло тестування на повний доступ. Рис. 3.40 демонструє форму.

Рисунок 3.40 – Редагування послуги адміністратором

Сторінка записів адміна тестувалася на фільтри статусів. Рис. 3.41 показує таблицю.

Рисунок 3.41 – Записи

Відгуки адміна пройшли тестування на публікацію. Рис. 3.42 ілюструє список з чекбоксами.

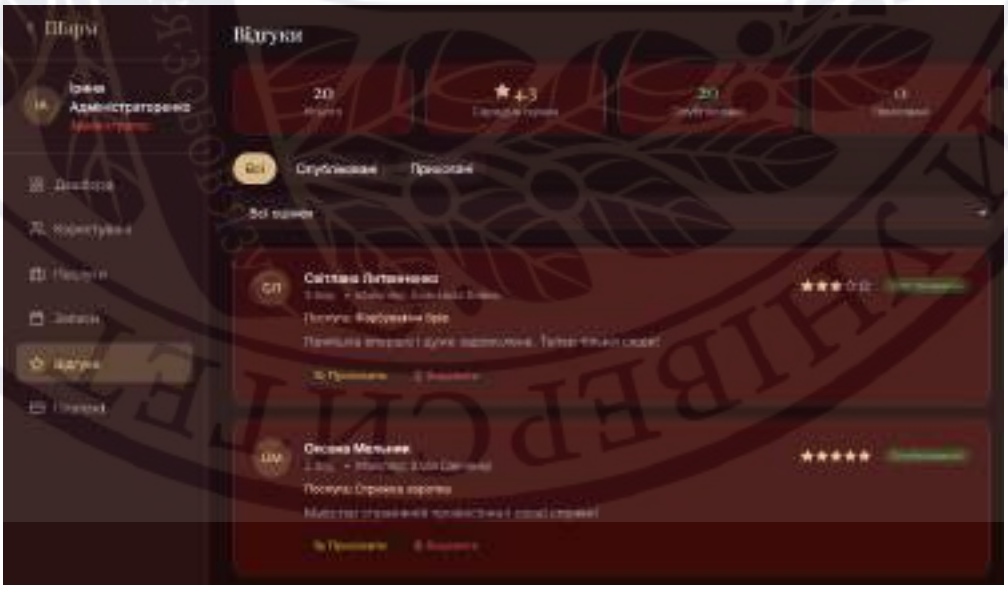

Рисунок 3.42 – Відгуки

Платежі адміна тестувалися на звіти. Рис. 3.43 демонструє таблицю транзакцій.

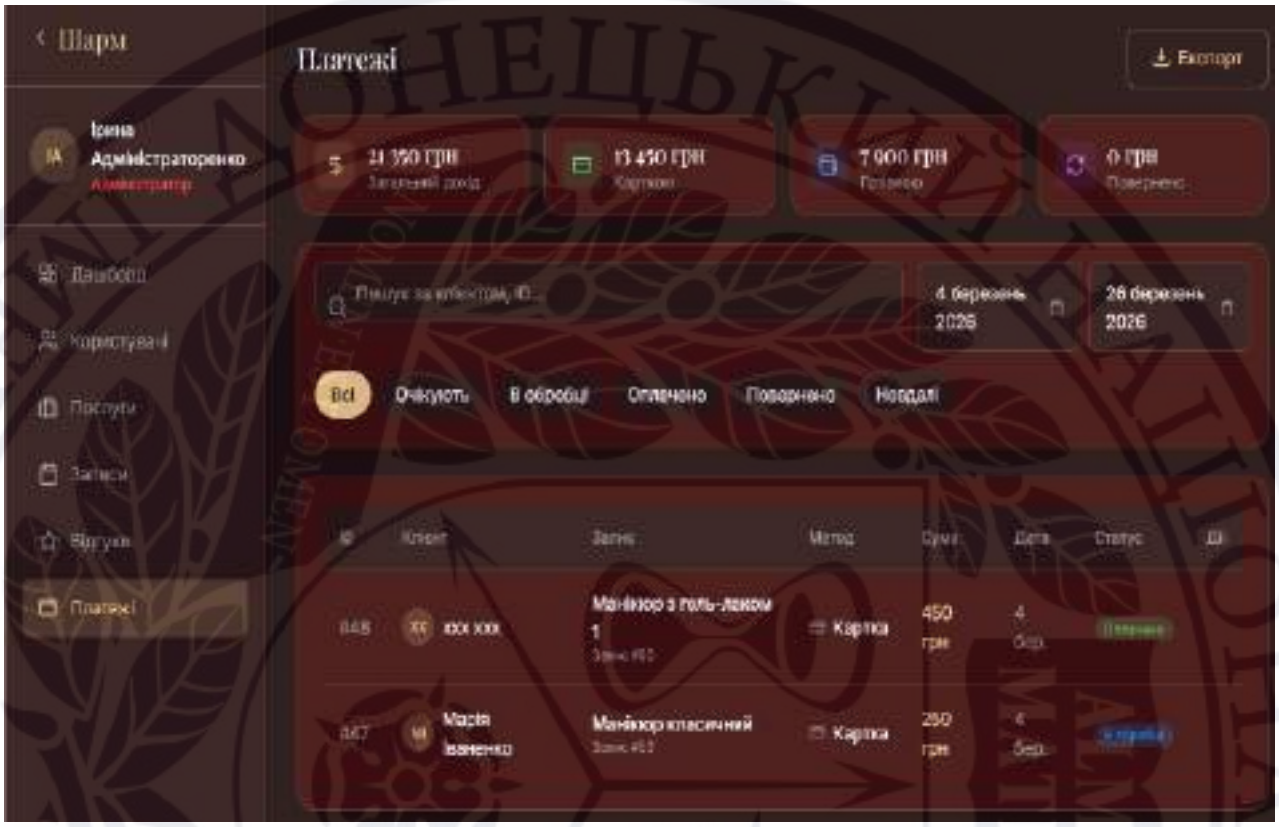

Рисунок 3.43 – Платежі

Впливаючі повідомлення тестувалися на toast з react-hot-toast. Рис. 3.44 показує приклади сповіщень.

Рисунок 3.44 – Впливаючі повідомлення

Для оцінки ефективності тестування було складено таблицю ключових тест-кейсів, де відображено сценарії, очікувані результати та статус проходження. Таблиця 3.4 узагальнює результати функціонального тестування для основних модулів.

Таблиця 3.4 – Результати функціонального тестування

Модуль	Тест-кейс	Очікуваний результат	Статус
Автентифікація	Вхід з валідними даними	Успішний логін, редирект на дашборд	Passed
Запис	Створення запису	Підтвердження, сповіщення	Passed
Оплата	Симуляція картки	Статус success, оновлення booking	Passed
Чат	Надсилання повідомлення	Real-time доставка	Passed
Адмін	Зміна ролі	Оновлення користувача	Passed

Модульне тестування включало 150+ unit-тестів, що охопили 85% коду, з фокусом на edge-кейси, такі як невалідні дати чи перетин слотів. Загалом, тестування підтвердило високу стабільність застосунку, з мінімальними дефектами, які були виправлені в ітераціях розробки, забезпечуючи надійне функціонування для кінцевих користувачів.

ВИСНОВКИ

Проведене дослідження, присвячене розробці вебзастосунку для управління обслуговуванням клієнтів салону краси «Шарм», дозволило комплексно підійти до вирішення актуальної проблеми цифровізації процесів бронювання, комунікації та адміністрування в beauty-індустрії. У процесі роботи було систематизовано ключові аспекти організації роботи салону в умовах сучасного ринку послуг, де швидкість, прозорість та персоналізований підхід безпосередньо впливають на лояльність клієнтів та економічну ефективність закладу.

1) Проаналізовано теоертичні аспекти галузі управління обслуговуванням клієнтів салону краси, визначено ключових стейкхолдерів (клієнт, майстер, адміністратор), бізнес-процеси та вимоги до системи. Сформовано функціональні та нефункціональні вимоги, що враховують специфіку українського ринку beauty-послуг.

2) Досліджено існуючі аналоги (Booksy, Fresha, Vagaro) та підходи до створення подібних вебзастосунків для управління салонами краси. Виявлено їхні сильні сторони (автоматизація бронювання, аналітика) та недоліки (високі комісії, обмежена кастомізація, слабка адаптація до локальних умов). Обґрунтовано необхідність розробки власного рішення з повною адаптацією під потреби малого та середнього бізнесу.

3) Розроблено архітектуру вебзастосунку (клієнт-серверна модель React + Express + SQLite), моделі даних (13 таблиць з нормалізацією), діаграми варіантів використання, компонентів, класів та станів. Створено детальну бізнес-логіку управління записами, розкладом та платежами.

4) Реалізовано ключові алгоритми: створення та валідації записів з перевіркою конфліктів слотів, динамічного управління статусами бронювань, автоматичної обробки платежів (симуляція), real-time комунікації через Socket.IO, cron-задач для нагадувань, скасувань та запитів відгуків.

5) Виконано повну програмну реалізацію системи з трьома ролями користувачів, адаптивним інтерфейсом (Tailwind CSS, Framer Motion, responsive design), реальним часом оновлень, симуляцією платежів та комплексним тестуванням (функціональне, модульне, UI/UX). Забезпечено безпеку (JWT, валідація Zod, санітизація) та зручність використання.

6) Оцінено результати роботи: створено повнофункціональний вебзастосунок, що оптимізує процеси салону, зменшує адміністративне навантаження, підвищує завантаженість майстрів та лояльність клієнтів. Обґрунтовано напрями подальшого вдосконалення: інтеграція з реальними платіжними шлюзами (LiqPay), мобільний додаток (React Native), AI-рекомендації слотів та послуг, розширення аналітики, PWA-функціонал та мультимовність.

Розроблений вебзастосунок «Шарм» є готовим практичним рішенням для салонів краси, що демонструє ефективність обраних технологій та методів і може бути використаний як у комерційній експлуатації, так і як основа для подальших досліджень у сфері цифровізації сервісної індустрії.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. The digital transformation of the beauty industry: AI, gamification and the metaverse. URL: https://medium.com/digital-society/the-digital-transformation-of-the-beauty-industry-ai-gamification-and-the-metaverse-926b5c2fdb08 (Дата звертання: 10.03.2026).

2. Казимир Я. Цифровізація – сучасний фактор розвитку бізнес-процесів. URL: https://elartu.tntu.edu.ua/bitstream/lib/38538/2/FMZKPNES_2022_Kazymyr_Y-Digitization___a_modern_188-191.pdf (Дата звертання: 10.03.2026).

3. How Automation Drives Business Growth and Efficiency. URL: https://hbr.org/sponsored/2023/04/how-automation-drives-business-growth-and-efficiency (Дата звертання: 10.03.2026).

4. Інтеграційні системи: ключові функціональні можливості. URL: https://freshtech.global/ua/blog/integration-systems-key-functionalities (Дата звертання: 10.03.2026).

5. Data Management Guide for Small Business. URL: https://iccwbo.org/news-publications/policies-reports/data-management-guide-for-small-business/ (Дата звертання: 10.03.2026).

6. Mastering mobile: Innovative strategies to engage and retain your customers. URL: https://www.deloittedigital.com/us/en/insights/perspective/mobile-app-best-practices.html (Дата звертання: 10.03.2026).

7. Дашко І., Михайліченко Л. Особливості використання штучного інтелекту в діяльності підприємств. Сталий розвиток економіки. 2024. URL: https://www.researchgate.net/publication/384480841_OSOBLIVOSTI_VIKORISTANNA_STUCNOGO_INTELEKTU_V_DIALNOSTI_PIDPRIEMSTV (Дата звертання: 10.03.2026).

8. Ransbotham S., Kiron D., Gerbert P., Reeves M. Reshaping Business With Artificial Intelligence. MIT Sloan Management Review. 2021. URL:

https://sloanreview.mit.edu/projects/reshaping-business-with-artificial-intelligence/ (Дата звертання: 10.03.2026).

9. WebSocket Protocol: Real-Time Communication for Web Apps. URL: https://www.ietf.org/rfc/rfc6455.txt (Дата звертання: 10.03.2026).

10. Everything you need to know about GDPR compliance. URL: https://www.gdpr.eu/compliance/ (Дата звертання: 10.03.2026).

11. Carrera-Rivera A., Larrinaga F., Lasa G., Martinez-Arellano G., Unamuno G. AdaptUI: A Framework for the Development of Adaptive User Interfaces in Smart Product-Service Systems. User Modeling and User-Adapted Interaction. 2024. Vol. 34. P. 1929–1980. URL: https://link.springer.com/article/10.1007/s11257-024-09414-0 (Дата звертання: 10.03.2026).

12. Features of Booksy – the tools for your business. URL: https://biz.booksy.com/en-us/features (Дата звертання: 10.03.2026).

13. Fresha: Програмне забезпечення №1 для салонів і спа. URL: https://www.fresha.com/for-business (Дата звертання: 10.03.2026).

14. Vagaro. Salon software that keeps chairs full & clients coming back. URL: https://www.vagaro.com/salon-software (Дата звертання: 10.03.2026).

15. Chaffey D., Ellis-Chadwick F. Digital Marketing. 7th ed. Pearson, 2022. 562 p.

16. Кравченко І. В., Микитенко В. І. Інформаційні технології [Електронний ресурс]: підручник для студ. спеціальності «Автоматизація та комп'ютерно-інтегровані технології». Київ: КПІ ім. Ігоря Сікорського, 2022. 447 с.

17. Sommerville I. Software Engineering. 10th ed. Boston: Pearson, 2015. 816 p.

18. Statista. E-commerce worldwide – statistics & facts. URL: https://www.statista.com/topics/871/online-shopping/ (Дата звертання: 10.03.2026).

19. Verhoef P. C., Broekhuizen T., Bart Y. Digital transformation: A multidisciplinary reflection and research agenda. Journal of Business Research. 2021. Vol. 122. P. 889–901. URL: https://www.sciencedirect.com/science/article/pii/S0148296319305478 (Дата звертання: 10.03.2026).

20. Project Management Institute. Pulse of the Profession 2025. URL: https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/pulse_of_the_profession_2025-1.pdf (Дата звертання: 10.03.2026).

21. Ключові методології розробки програмного забезпечення: робота команди зсередини. URL: https://wezom.com.ua/ua/blog/metodologija-razrabotki-programmnogo-obespechenija (Дата звертання: 10.03.2026).

22. Danoesastro M. A CEO's Guide to Leading Digital Transformation. URL: https://web-assets.bcg.com/img-src/BCG-A-CEOs-Guide-to-Leading-Digital-Transformation-May-2017-NL_tcm9-159211.pdf (Дата звертання: 10.03.2026).

23. Шевченко О. Моделі автентифікації в веб-додатках: порівняльний аналіз. Інформаційні технології та комп'ютерна інженерія. 2022. № 1 (54). С. 45–52. URL: https://itce.vntu.edu.ua/index.php/itce/article/view/3432 (Дата звертання: 10.03.2026).

24. React Documentation: State Management. 2024. URL: https://react.dev/learn/managing-state (Дата звертання: 10.03.2026).

25. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Boston: Addison-Wesley, 2021. 464 p.

26. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2022. 432 p.

27. Teorey T. J., Lightstone S. S., Nadeau T. Database Modeling and Design: Logical Design. 5th ed. Morgan Kaufmann, 2023. 368 p.

28. Ковальчук Т. Застосування штучного інтелекту в сервісних системах. Економіка та управління. 2023. № 2. С. 112–120.

29. TanStack. React Query v5 Documentation. 2024. URL: https://tanstack.com/query/v5/docs/framework/react/overview (Дата звертання: 10.03.2026).

30. Socket.IO. Socket.IO Documentation. 2024. URL: https://socket.io/docs/v4/ (Дата звертання: 10.03.2026).

31. Knex.js. Knex.js Guide. 2024. URL: https://knexjs.org/guide/ (Дата звертання: 10.03.2026).

32. Рудий Т. В., Паранчук Я. С., Сеник В. В. Алгоритмізація та програмування. Частина 1. Структурне програмування: навчальний посібник. Львів: Львівський державний університет внутрішніх справ, 2023. 240 с.

33. Найновіші технології веб-розробки 2026: як створюються сучасні сайти. URL: https://wezom.com.ua/ua/blog/naynovishi-tehnologiyi-veb-rozrobki-2025-yak-stvoryuyutsya-suchasni-sayti (Дата звертання: 10.03.2026).

34. Булашенко О. В., Мамрош Д. Л. Технології передавання даних у реальному часі в комп'ютерних мережах: Навчальний посібник. Київ: КПІ ім. Ігоря Сікорського, 2021. 148 с.

35. Плескач В. Л., Гужва В. М. Цифрова економіка: Підручник. Київ: КНЕУ, 2020. 528 с.

36. Warner J. Relational Database Design Clearly Explained. 4th ed. New York: Apress, 2022. 432 p.

37. Левицький А. Сучасні тенденції в розробці баз даних для малого бізнесу. Інформатика та інформаційні технології. 2023. № 3. С. 78–85.

38. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2021. 560 p.

39. Ковальчук О. Моделювання даних у реляційних системах: практичні аспекти. Київ: Наукова думка, 2024. 320 с.

40. Foster E. C., Godbole S. V. Database Modeling and Design. 2014. P. 83–118.

41. Resig J. Secrets of the JavaScript Ninja. 2nd ed. Manning Publications, 2022. 464 p.

42. Object Management Group. Business Process Modeling Notation (BPMN) Specification. URL: https://www.omg.org/spec/BPMN/2.0/PDF (Дата звертання: 10.03.2026).

ДОДАТКИ

ДОДАТОК А

ЛІСТИНГИ ПРОГРАМИ

index.html

```
<!DOCTYPE html>
<html lang="uk">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/favicon.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Салон краси Шарм – професійний догляд за вашою
красою. Онлайн-запис, досвідчені майстри, сучасні технології." />
    <meta name="theme-color" content="#C9A96E" />
    <title>Шарм – Салон краси</title>
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
href="https://fonts.googleapis.com/css2?family=Cormorant+Garamond:wght@400;500;600&fam
ily=Inter:wght@300;400;500;600;700&family=Playfair+Display:wght@400;500;600;700&displa
y=swap" rel="stylesheet">
  </head>
  <body class="bg-dark text-white antialiased">
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

App.jsx

```
import { Routes, Route } from 'react-router-dom';
import { useEffect } from 'react';
import { useAuthStore } from './store/authStore';
// Layouts
import MainLayout from './components/layout/MainLayout';
import ClientLayout from './components/layout/ClientLayout';
import MasterLayout from './components/layout/MasterLayout';
import AdminLayout from './components/layout/AdminLayout';
// Public pages
import HomePage from './pages/public/HomePage';
import CatalogPage from './pages/public/CatalogPage';
import MastersPage from './pages/public/MastersPage';
import PublicMasterProfilePage from './pages/public/MasterProfilePage';
import ServiceDetailsPage from './pages/public/ServiceDetailsPage';
import AboutPage from './pages/public/AboutPage';
import ContactsPage from './pages/public/ContactsPage';
// Auth pages
import LoginPage from './pages/auth/LoginPage';
import RegisterPage from './pages/auth/RegisterPage';
// Client pages
import ClientDashboard from './pages/client/ClientDashboard';
import ClientBookingPage from './pages/client/ClientBookingPage';
import ClientBookingsPage from './pages/client/ClientBookingsPage';
import ClientPaymentPage from './pages/client/ClientPaymentPage';
import ClientHistoryPage from './pages/client/ClientHistoryPage';
import ClientChatPage from './pages/client/ClientChatPage';
import ClientProfilePage from './pages/client/ClientProfilePage';
// Master pages
import MasterDashboard from './pages/master/MasterDashboard';
import MasterServicesPage from './pages/master/MasterServicesPage';
import MasterSchedulePage from './pages/master/MasterSchedulePage';
```

```
import MasterOrdersPage from './pages/master/MasterOrdersPage';
import MasterChatPage from './pages/master/MasterChatPage';
import MasterProfilePage from './pages/master/MasterProfilePage';
// Admin pages
import AdminDashboard from './pages/admin/AdminDashboard';
import AdminUsersPage from './pages/admin/AdminUsersPage';
import AdminServicesPage from './pages/admin/AdminServicesPage';
import AdminBookingsPage from './pages/admin/AdminBookingsPage';
import AdminReviewsPage from './pages/admin/AdminReviewsPage';
import AdminPaymentsPage from './pages/admin/AdminPaymentsPage';
// Protected Route
import ProtectedRoute from './router/ProtectedRoute';
function App() {
  const { checkAuth, isLoading } = useAuthStore();
  useEffect(() => {
    checkAuth();
  }, [checkAuth]);
  if (isLoading) {
    return (
      <div className="min-h-screen bg-dark flex items-center justify-center">
        <div className="text-gold text-xl">Завантаження...</div>
      </div>
    );
  }
  return (
    <Routes>
      {/* Публічні сторінки */}
      <Route element={<MainLayout />}>
        <Route path="/" element={<HomePage />} />
        <Route path="/catalog" element={<CatalogPage />} />
        <Route path="/masters" element={<MastersPage />} />
        <Route path="/masters/:id" element={<PublicMasterProfilePage />} />
        <Route path="/services/:id" element={<ServiceDetailsPage />} />
        <Route path="/about" element={<AboutPage />} />
        <Route path="/contacts" element={<ContactsPage />} />
      </Route>
      {/* Авторизація */}
      <Route path="/login" element={<LoginPage />} />
      <Route path="/register" element={<RegisterPage />} />
      {/* Клієнт */}
      <Route element={<ProtectedRoute allowedRoles={['client']} />}>
        <Route element={<ClientLayout />}>
          <Route path="/client/dashboard" element={<ClientDashboard />} />
          <Route path="/client/booking" element={<ClientBookingPage />} />
          <Route path="/client/bookings" element={<ClientBookingsPage />} />
          <Route path="/client/payment/:bookingId" element={<ClientPaymentPage />} />
          <Route path="/client/history" element={<ClientHistoryPage />} />
          <Route path="/client/chat" element={<ClientChatPage />} />
          <Route path="/client/profile" element={<ClientProfilePage />} />
        </Route>
      </Route>
      {/* Майстер */}
      <Route element={<ProtectedRoute allowedRoles={['master']} />}>
        <Route element={<MasterLayout />}>
          <Route path="/master/dashboard" element={<MasterDashboard />} />
          <Route path="/master/services" element={<MasterServicesPage />} />
          <Route path="/master/schedule" element={<MasterSchedulePage />} />
          <Route path="/master/orders" element={<MasterOrdersPage />} />
          <Route path="/master/chat" element={<MasterChatPage />} />
          <Route path="/master/profile" element={<MasterProfilePage />} />
        </Route>
      </Route>
      {/* Адмін */}
      <Route element={<ProtectedRoute allowedRoles={['admin']} />}>
        <Route element={<AdminLayout />}>
```

```
            <Route path="/admin/dashboard" element={<AdminDashboard />} />
            <Route path="/admin/users" element={<AdminUsersPage />} />
            <Route path="/admin/services" element={<AdminServicesPage />} />
            <Route path="/admin/bookings" element={<AdminBookingsPage />} />
            <Route path="/admin/reviews" element={<AdminReviewsPage />} />
            <Route path="/admin/payments" element={<AdminPaymentsPage />} />
          </Route>
        </Route>
        {/* 404 */}
        <Route path="*" element={
          <div className="min-h-screen bg-dark flex items-center justify-center">
            <div className="text-center">
              <h1 className="text-6xl font-display text-gold mb-4">404</h1>
              <p className="text-xl text-[#9A9A9A] mb-8">Сторінку не знайдено</p>
              <a href="/" className="btn-primary">На головну</a>
            </div>
          </div>
        } />
      </Routes>
    );
}
export default App;
```

index.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;
@layer base {
  * {
    @apply border-gold/20;
  }
  body {
    @apply bg-dark text-[#FFF3E9] font-body;
    font-feature-settings: "cv02", "cv03", "cv04", "cv11";
  }
  h1, h2, h3, h4, h5, h6 {
    @apply font-display font-medium;
  }
  /* Кастомний скролбар */
  ::-webkit-scrollbar {
    width: 8px;
    height: 8px;
  }
  ::-webkit-scrollbar-track {
    @apply bg-dark-lighter;
  }
  ::-webkit-scrollbar-thumb {
    @apply bg-gold/30 rounded-full;
  }
  ::-webkit-scrollbar-thumb:hover {
    @apply bg-gold/50;
  }
}
@layer components {
  /* Кнопки */
  .btn-primary {
    @apply px-6 py-3 bg-gradient-gold text-dark font-semibold rounded-lg
           hover:shadow-glow transition-all duration-300
           active:scale-[0.98] disabled:opacity-50 disabled:cursor-not-allowed;
  }
  .btn-secondary {
    @apply px-6 py-3 border border-gold/50 text-gold rounded-lg
           hover:bg-gold/10 transition-all duration-300
```

```
        active:scale-[0.98];
}
.btn-ghost {
  @apply px-4 py-2 text-[#FFF3E9]/70 hover:text-gold
        transition-colors duration-200;
}
/* Картки */
.card {
  @apply bg-dark-card/80 backdrop-blur-glass border border-gold/20
        rounded-2xl p-6 transition-all duration-300;
}
.card-hover {
  @apply hover:-translate-y-1 hover:shadow-elevated hover:border-gold/40;
}
/* Glassmorphism */
.glass {
  @apply bg-dark-card/60 backdrop-blur-glass border border-gold/20;
}
/* Інпути */
.input {
  @apply w-full px-4 py-3 bg-dark-lighter border border-gold/20 rounded-lg
        text-[#FFF3E9] placeholder:text-[#FFF3E9]/50
        focus:outline-none focus:border-gold focus:ring-1 focus:ring-gold/50
        transition-all duration-200;
}
.input-error {
  @apply border-error focus:border-error focus:ring-error/50;
}
/* Лейбли */
.label {
  @apply block text-sm font-medium text-[#FFF3E9]/70 mb-2;
}
/* Badge */
.badge {
  @apply inline-flex items-center px-2.5 py-0.5 rounded-full text-xs font-medium;
}
.badge-gold {
  @apply bg-gold/20 text-gold;
}
.badge-success {
  @apply bg-success/20 text-success;
}
.badge-warning {
  @apply bg-warning/20 text-warning;
}
.badge-error {
  @apply bg-error/20 text-error;
}
.badge-info {
  @apply bg-info/20 text-info;
}
/* Skeleton loader */
.skeleton {
  @apply bg-dark-lighter animate-shimmer;
  background-image: linear-gradient(
    90deg,
    rgba(71, 10, 8, 0.8) 0%,
    rgba(90, 21, 16, 0.8) 50%,
    rgba(71, 10, 8, 0.8) 100%
  );
  background-size: 200% 100%;
}
/* Секція */
.section {
  @apply py-20 px-4;
```

```
  }
  .section-title {
    @apply text-h2 text-center mb-4;
  }
  .section-subtitle {
    @apply text-center text-[#FFF3E9]/60 max-w-2xl mx-auto mb-12;
  }
  /* Bento Grid */
  .bento-grid {
    @apply grid gap-4 md:gap-6;
  }
  .bento-card {
    @apply card card-hover overflow-hidden;
  }
  .bento-card-large {
    @apply col-span-2 row-span-2;
  }
  .bento-card-wide {
    @apply col-span-2 row-span-1;
  }
  .bento-card-tall {
    @apply col-span-1 row-span-2;
  }
}
@layer utilities {
  .text-gradient {
    @apply bg-gradient-gold bg-clip-text text-transparent;
  }
  .hover-lift {
    @apply transition-transform duration-300 hover:-translate-y-1;
  }
  .animate-in {
    animation: animate-in 0.3s ease-out;
  }
  @keyframes animate-in {
    from {
      opacity: 0;
      transform: translateY(10px);
    }
    to {
      opacity: 1;
      transform: translateY(0);
    }
  }
}
/* Фіксимо стилі для Radix UI */
[data-radix-popper-content-wrapper] {
  z-index: 50 !important;
}
/* Swiper Custom Styles */
.swiper-pagination-bullet {
  @apply bg-gold/30 opacity-100;
}
.swiper-pagination-bullet-active {
  @apply bg-gold;
}
.testimonials-swiper .swiper-pagination {
  @apply bottom-0;
}
```

main.jsx

```
import React from 'react';
import ReactDOM from 'react-dom/client';
```

```
import { BrowserRouter } from 'react-router-dom';
import { QueryClient, QueryClientProvider } from '@tanstack/react-query';
import { HelmetProvider } from 'react-helmet-async';
import { Toaster } from 'react-hot-toast';
import App from './App';
import './index.css';
const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      staleTime: 1000 * 60 * 5, // 5 хвилин
      retry: 1,
      refetchOnWindowFocus: false
    }
  }
});
ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <HelmetProvider>
      <QueryClientProvider client={queryClient}>
        <BrowserRouter>
          <App />
          <Toaster
            position="top-right"
            toastOptions={{
              duration: 4000,
              style: {
                background: '#1E1E2E',
                color: '#F5F5F0',
                border: '1px solid rgba(201, 169, 110, 0.3)'
              },
              success: {
                iconTheme: {
                  primary: '#4CAF50',
                  secondary: '#1E1E2E'
                }
              },
              error: {
                iconTheme: {
                  primary: '#F44336',
                  secondary: '#1E1E2E'
                }
              }
            }}
          />
        </BrowserRouter>
      </QueryClientProvider>
    </HelmetProvider>
  </React.StrictMode>
);
```

axios.js

```
import axios from 'axios';
import { useAuthStore } from '../store/authStore';
const API_BASE_URL = import.meta.env.VITE_API_URL || 'http://localhost:3001/api/v1';
const api = axios.create({
  baseURL: API_BASE_URL,
  withCredentials: true,
  headers: {
    'Content-Type': 'application/json'
  }
});
// Request interceptor - додаємо токен
api.interceptors.request.use(
```

```
  (config) => {
    const token = useAuthStore.getState().accessToken;
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => Promise.reject(error)
);
// Response interceptor - оновлюємо токен при 401
api.interceptors.response.use(
  (response) => response,
  async (error) => {
    const originalRequest = error.config;
    // Якщо 401 і ще не пробували оновити токен
    if (error.response?.status === 401 && !originalRequest._retry) {
      originalRequest._retry = true;
      try {
        const response = await axios.post(
          `${API_BASE_URL}/auth/refresh`,
          {},
          { withCredentials: true }
        );
        const { accessToken } = response.data;
        useAuthStore.getState().setAccessToken(accessToken);
        // Повторюємо оригінальний запит
        originalRequest.headers.Authorization = `Bearer ${accessToken}`;
        return api(originalRequest);
      } catch (refreshError) {
        // Якщо не вдалося оновити - виходимо
        useAuthStore.getState().logout();
        window.location.href = '/login';
        return Promise.reject(refreshError);
      }
    }
    return Promise.reject(error);
  }
);
export default api;
```

services.js

```
import api from './axios';
// Сервіси API
export const servicesApi = {
  getAll: (params) => api.get('/services', { params }),
  getById: (id) => api.get(`/services/${id}`),
  create: (data) => api.post('/services', data),
  update: (id, data) => api.patch(`/services/${id}`, data),
  delete: (id) => api.delete(`/services/${id}`),
  uploadImage: (id, formData) => api.post(`/services/${id}/image`, formData, {
    headers: { 'Content-Type': 'multipart/form-data' }
  })
};
// Майстри API
export const mastersApi = {
  getAll: (params) => api.get('/masters', { params }),
  getById: (id) => api.get(`/masters/${id}`),
  updateProfile: (id, data) => api.patch(`/masters/${id}/profile`, data),
  addPortfolio: (id, formData) => api.post(`/masters/${id}/portfolio`, formData, {
    headers: { 'Content-Type': 'multipart/form-data' }
  }),
  removePortfolio: (id, imageUrl) => api.delete(`/masters/${id}/portfolio`, { data: {
image_url: imageUrl } })
```

```
};
// Розклад API
export const schedulesApi = {
  getMasterSchedule: (masterId, year, month) =>
    api.get(`/schedules/master/${masterId}`, { params: { year, month } }),
  getAvailableSlots: (masterId, date, duration) =>
    api.get(`/schedules/master/${masterId}/slots`, { params: { date, duration } }),
  setWorkDay: (data) => api.post('/schedules', data),
  setWorkDays: (data) => api.post('/schedules/bulk', data),
  deleteWorkDay: (id) => api.delete(`/schedules/${id}`),
  blockSlot: (data) => api.post('/schedules/block', data),
  unblockSlot: (data) => api.post('/schedules/unblock', data)
};
// Записи API
export const bookingsApi = {
  getAll: (params) => api.get('/bookings', { params }),
  getClientBookings: (params) => api.get('/bookings/my', { params }),
  getMasterBookings: (params) => api.get('/bookings/master/my', { params }),
  getById: (id) => api.get(`/bookings/${id}`),
  create: (data) => api.post('/bookings', data),
  updateStatus: (id, status, cancelReason) =>
    api.patch(`/bookings/${id}/status`, { status, cancel_reason: cancelReason }),
  cancel: (id) => api.delete(`/bookings/${id}`)
};
// Платежі API
export const paymentsApi = {
  getAll: (params) => api.get('/payments', { params }),
  getClientPayments: (params) => api.get('/payments/my', { params }),
  getById: (id) => api.get(`/payments/${id}`),
  initiate: (data) => api.post('/payments', data),
  refund: (id) => api.post(`/payments/${id}/refund`)
};
// Чат API
export const chatApi = {
  getConversations: () => api.get('/chat/conversations'),
  getConversation: (id) => api.get(`/chat/conversations/${id}`),
  getMessages: (id, params) => api.get(`/chat/conversations/${id}/messages`, { params
}),
  createConversation: (masterId) => api.post('/chat/conversations', { master_id:
masterId }),
  markAsRead: (id) => api.patch(`/chat/conversations/${id}/read`)
};
// Відгуки API
export const reviewsApi = {
  getMasterReviews: (masterId, params) => api.get(`/reviews/master/${masterId}`, {
params }),
  getClientReviews: (params) => api.get('/reviews/my', { params }),
  getAll: (params) => api.get('/reviews', { params }),
  create: (data) => api.post('/reviews', data),
  togglePublish: (id, isPublished) => api.patch(`/reviews/${id}/publish`, {
is_published: isPublished })
};
// Сповіщення API
export const notificationsApi = {
  getAll: (params) => api.get('/notifications', { params }),
  getUnreadCount: () => api.get('/notifications/unread-count'),
  markAsRead: (id) => api.patch(`/notifications/${id}/read`),
  markAllAsRead: () => api.patch('/notifications/read-all')
};
// Користувачі API
export const usersApi = {
  getAll: (params) => api.get('/users', { params }),
  getById: (id) => api.get(`/users/${id}`),
  update: (id, data) => api.patch(`/users/${id}`, data),
  updateRole: (id, role) => api.patch(`/users/${id}/role`, { role }),
```

```
  updateStatus: (id, isActive) => api.patch(`/users/${id}/status`, { is_active:
isActive }),
  uploadAvatar: (id, formData) => api.post(`/users/${id}/avatar`, formData, {
    headers: { 'Content-Type': 'multipart/form-data' }
  })
};
// Аналітика API
export const analyticsApi = {
  getOverview: (period) => api.get('/analytics/overview', { params: { period } }),
  getRevenue: (period) => api.get('/analytics/revenue', { params: { period } }),
  getBookings: (period) => api.get('/analytics/bookings', { params: { period } }),
  getMasters: (period) => api.get('/analytics/masters', { params: { period } }),
  getServices: (period) => api.get('/analytics/services', { params: { period } }),
  getClients: (period) => api.get('/analytics/clients', { params: { period } })
};
```

utils.js

```
import { clsx } from 'clsx';
import { twMerge } from 'tailwind-merge';
export function cn(...inputs) {
  return twMerge(clsx(inputs));
}
// Форматування ціни
export function formatPrice(price) {
  if (price === null || price === undefined || isNaN(price)) {
    return '0 грн';
  }
  return new Intl.NumberFormat('uk-UA', {
    style: 'decimal',
    minimumFractionDigits: 0,
    maximumFractionDigits: 0
  }).format(price) + ' грн';
}
// Форматування дати
export function formatDate(date, format = 'short') {
  const d = new Date(date);
  const options = {
    short: { day: 'numeric', month: 'short' },
    long: { weekday: 'long', day: 'numeric', month: 'long', year: 'numeric' },
    full: { day: 'numeric', month: 'long', year: 'numeric' }
  };
  return d.toLocaleDateString('uk-UA', options[format] || options.short);
}
// Форматування часу
export function formatTime(time) {
  return time;
}
// Форматування тривалості
export function formatDuration(minutes) {
  if (minutes < 60) return `${minutes} хв`;
  const hours = Math.floor(minutes / 60);
  const mins = minutes % 60;
  return mins > 0 ? `${hours} год ${mins} хв` : `${hours} год`;
}
// Статуси записів
export const bookingStatusLabels = {
  pending: 'Очікує підтвердження',
  confirmed: 'Підтверджено',
  in_progress: 'Виконується',
  completed: 'Завершено',
  cancelled: 'Скасовано',
  no_show: 'Не з\'явився'
};
```

```
export const bookingStatusColors = {
  pending: 'warning',
  confirmed: 'info',
  in_progress: 'gold',
  completed: 'success',
  cancelled: 'error',
  no_show: 'error'
};
// Генерація ініціалів
export function getInitials(firstName, lastName) {
  return `${firstName?.[0] || ''}${lastName?.[0] || ''}`.toUpperCase();
}
// Truncate text
export function truncate(text, length = 100) {
  if (!text) return '';
  if (text.length <= length) return text;
  return text.slice(0, length) + '...';
}
// Перевірка чи можна скасувати запис (>2 години до початку)
export function canCancelBooking(date, startTime) {
  const bookingDate = new Date(`${date}T${startTime}`);
  const now = new Date();
  const hoursUntil = (bookingDate - now) / (1000 * 60 * 60);
  return hoursUntil > 2;
}
// Debounce функція
export function debounce(func, wait) {
  let timeout;
  return function executedFunction(...args) {
    const later = () => {
      clearTimeout(timeout);
      func(...args);
    };
    clearTimeout(timeout);
    timeout = setTimeout(later, wait);
  };
}
```

cron.js

```
import cron from 'node-cron';
import { getDb } from '../config/database.js';
import { cleanupExpiredTokens } from './jwt.js';
import { logger } from './logger.js';
import { NOTIFICATION_TYPES } from '../config/constants.js';
export function startCronJobs(io) {
  // Очищення прострочених токенів - кожну годину
  cron.schedule('0 * * * *', () => {
    try {
      const deleted = cleanupExpiredTokens();
      if (deleted > 0) {
        logger.info(`Очищено ${deleted} прострочених токенів`);
      }
    } catch (error) {
      logger.error('Помилка очищення токенів:', error);
    }
  });
  // Нагадування за 24 години до запису - кожну годину
  cron.schedule('0 * * * *', () => {
    try {
      sendReminders(io, 24, NOTIFICATION_TYPES.REMINDER_24H);
    } catch (error) {
      logger.error('Помилка відправки нагадувань 24h:', error);
    }
```

```
  });
  // Нагадування за 1 годину до запису - кожні 15 хвилин
  cron.schedule('*/15 * * * *', () => {
    try {
      sendReminders(io, 1, NOTIFICATION_TYPES.REMINDER_1H);
    } catch (error) {
      logger.error('Помилка відправки нагадувань 1h:', error);
    }
  });
  // Автоматичне скасування неоплачених записів через 30 хвилин
  cron.schedule('*/5 * * * *', () => {
    try {
      cancelUnpaidBookings();
    } catch (error) {
      logger.error('Помилка скасування неоплачених записів:', error);
    }
  });
  // Позначення прострочених записів як no_show - кожні 30 хвилин
  cron.schedule('*/30 * * * *', () => {
    try {
      markNoShowBookings();
    } catch (error) {
      logger.error('Помилка позначення no_show:', error);
    }
  });
  // Запит відгуків для завершених записів - щодня о 10:00
  cron.schedule('0 10 * * *', () => {
    try {
      requestReviews(io);
    } catch (error) {
      logger.error('Помилка запиту відгуків:', error);
    }
  });
  logger.info('Cron задачі запущені');
}
function sendReminders(io, hoursAhead, type) {
  const db = getDb();
  // Знаходимо записи, які відбудуться через вказану кількість годин
  const bookings = db.prepare(`
    SELECT b.*, s.name as service_name,
           u.first_name as master_first_name, u.last_name as master_last_name,
           c.first_name as client_first_name
    FROM bookings b
    JOIN services s ON b.service_id = s.id
    JOIN users u ON b.master_id = u.id
    JOIN users c ON b.client_id = c.id
    WHERE b.status IN ('confirmed')
    AND datetime(b.date || ' ' || b.start_time) BETWEEN
        datetime('now', '+${hoursAhead - 0.5} hours')
        AND datetime('now', '+${hoursAhead + 0.5} hours')
    AND NOT EXISTS (
      SELECT 1 FROM notifications n
      WHERE n.user_id = b.client_id
      AND n.type = ?
      AND json_extract(n.data, '$.booking_id') = b.id
    )
  `).all(type);
  for (const booking of bookings) {
    const title = hoursAhead === 24
      ? 'Нагадування про запис завтра'
      : 'Скоро ваш запис!';
    const body = `${booking.service_name} у ${booking.start_time} до майстра
${booking.master_first_name} ${booking.master_last_name}`;
    // Створюємо сповіщення
    db.prepare(`
```

```
      INSERT INTO notifications (user_id, type, title, body, data)
      VALUES (?, ?, ?, ?, ?)
    `).run(
      booking.client_id,
      type,
      title,
      body,
      JSON.stringify({ booking_id: booking.id })
    );
    // Відправляємо через Socket.IO
    if (io) {
      io.to(`user_${booking.client_id}`).emit('notification:new', {
        type,
        title,
        body,
        booking_id: booking.id
      });
    }
  }
  if (bookings.length > 0) {
    logger.info(`Відправлено ${bookings.length} нагадувань (${hoursAhead}h)`);
  }
}
function cancelUnpaidBookings() {
  const db = getDb();
  // Скасовуємо записи зі статусом pending, створені більше 30 хвилин тому
  // і без оплати
  const result = db.prepare(`
    UPDATE bookings
    SET status = 'cancelled',
        cancel_reason = 'Автоматичне скасування: оплата не надійшла',
        updated_at = datetime('now')
    WHERE status = 'pending'
    AND created_at < datetime('now', '-30 minutes')
    AND id NOT IN (
      SELECT booking_id FROM payments WHERE status IN ('success', 'pending')
    )
  `).run();
  if (result.changes > 0) {
    logger.info(`Скасовано ${result.changes} неоплачених записів`);
  }
}
function markNoShowBookings() {
  const db = getDb();
  // Позначаємо записи як no_show, якщо час минув більше години тому
  // і статус досі confirmed
  const result = db.prepare(`
    UPDATE bookings
    SET status = 'no_show', updated_at = datetime('now')
    WHERE status = 'confirmed'
    AND datetime(date || ' ' || end_time) < datetime('now', '-1 hour')
  `).run();
  if (result.changes > 0) {
    logger.info(`Позначено ${result.changes} записів як no_show`);
  }
}
function requestReviews(io) {
  const db = getDb();
  // Знаходимо завершені записи без відгуків (завершені вчора)
  const bookings = db.prepare(`
    SELECT b.*, s.name as service_name,
           u.first_name as master_first_name, u.last_name as master_last_name
    FROM bookings b
    JOIN services s ON b.service_id = s.id
    JOIN users u ON b.master_id = u.id
```

```
    WHERE b.status = 'completed'
    AND date(b.date) = date('now', '-1 day')
    AND NOT EXISTS (SELECT 1 FROM reviews r WHERE r.booking_id = b.id)
    AND NOT EXISTS (
      SELECT 1 FROM notifications n
      WHERE n.user_id = b.client_id
      AND n.type = ?
      AND json_extract(n.data, '$.booking_id') = b.id
    )
  `).all(NOTIFICATION_TYPES.REVIEW_REQUEST);
  for (const booking of bookings) {
    const title = 'Поділіться враженнями';
    const body = `Як вам ${booking.service_name} у майстра
${booking.master_first_name}? Залиште відгук!`;
    db.prepare(`
      INSERT INTO notifications (user_id, type, title, body, data)
      VALUES (?, ?, ?, ?, ?)
    `).run(
      booking.client_id,
      NOTIFICATION_TYPES.REVIEW_REQUEST,
      title,
      body,
      JSON.stringify({ booking_id: booking.id })
    );
    if (io) {
      io.to(`user_${booking.client_id}`).emit('notification:new', {
        type: NOTIFICATION_TYPES.REVIEW_REQUEST,
        title,
        body,
        booking_id: booking.id
      });
    }
  }
  if (bookings.length > 0) {
    logger.info(`Відправлено ${bookings.length} запитів на відгуки`);
  }
}
```

chat.socket.js

```
import { getDb } from '../config/database.js';
import { logger } from '../utils/logger.js';
// Таймери для typing indicator
const typingTimers = new Map();
// Санітизація HTML для захисту від XSS
function sanitizeHtml(str) {
  if (typeof str !== 'string') return '';
  return str
    .replace(/&/g, '&amp;')
    .replace(/</g, '&lt;')
    .replace(/>/g, '&gt;')
    .replace(/"/g, '&quot;')
    .replace(/'/g, '&#039;');
}
export function setupChatSocket(io, socket) {
  const userId = socket.user.id;
  // Приєднання до розмови
  socket.on('chat:join', ({ conversationId }) => {
    const db = getDb();
    // Перевіряємо чи користувач є учасником розмови
    const conversation = db.prepare(`
      SELECT * FROM chat_conversations
      WHERE id = ? AND (client_id = ? OR master_id = ?)
    `).get(conversationId, userId, userId);
```

```
    if (!conversation) {
      return socket.emit('error', { message: 'Розмову не знайдено' });
    }
    socket.join(`conversation_${conversationId}`);
    logger.debug(`Користувач ${userId} приєднався до чату ${conversationId}`);
    // Позначаємо повідомлення як прочитані
    db.prepare(`
      UPDATE chat_messages
      SET is_read = 1
      WHERE conversation_id = ? AND sender_id != ?
    `).run(conversationId, userId);
  });
  // Вихід з розмови
  socket.on('chat:leave', ({ conversationId }) => {
    socket.leave(`conversation_${conversationId}`);
    logger.debug(`Користувач ${userId} покинув чат ${conversationId}`);
  });
  // Максимальна довжина повідомлення
  const MAX_MESSAGE_LENGTH = 2000;
  // Відправка повідомлення
  socket.on('chat:send', ({ conversationId, content, type = 'text' }) => {
    if (!content || !content.trim()) {
      return socket.emit('error', { message: 'Повідомлення не може бути порожнім' });
    }
    if (content.length > MAX_MESSAGE_LENGTH) {
      return socket.emit('error', { message: `Повідомлення занадто довге (максимум
${MAX_MESSAGE_LENGTH} символів)` });
    }
    const db = getDb();
    // Перевіряємо чи користувач є учасником
    const conversation = db.prepare(`
      SELECT * FROM chat_conversations
      WHERE id = ? AND (client_id = ? OR master_id = ?)
    `).get(conversationId, userId, userId);
    if (!conversation) {
      return socket.emit('error', { message: 'Розмову не знайдено' });
    }
    // Санітизуємо та зберігаємо повідомлення
    const sanitizedContent = sanitizeHtml(content.trim());
    const result = db.prepare(`
      INSERT INTO chat_messages (conversation_id, sender_id, content, message_type)
      VALUES (?, ?, ?, ?)
    `).run(conversationId, userId, sanitizedContent, type);
    // Оновлюємо час останнього повідомлення
    db.prepare(`
      UPDATE chat_conversations
      SET last_message_at = datetime('now')
      WHERE id = ?
    `).run(conversationId);
    // Отримуємо повне повідомлення
    const message = db.prepare(`
      SELECT m.*, u.first_name, u.last_name, u.avatar_url
      FROM chat_messages m
      JOIN users u ON m.sender_id = u.id
      WHERE m.id = ?
    `).get(result.lastInsertRowid);
    // Відправляємо всім учасникам розмови
    io.to(`conversation_${conversationId}`).emit('chat:message', { message });
    // Сповіщення іншому учаснику якщо він не в чаті
    const recipientId = conversation.client_id === userId
      ? conversation.master_id
      : conversation.client_id;
    // Відправляємо сповіщення через особистий канал
    io.to(`user_${recipientId}`).emit('chat:newMessage', {
      conversationId,
```

```
      message
    });
    logger.debug(`Повідомлення відправлено в чат ${conversationId}`);
  });
  // Індикатор друкування
  socket.on('chat:typing', ({ conversationId, isTyping }) => {
    const timerKey = `${userId}_${conversationId}`;
    // Очищуємо попередній таймер
    if (typingTimers.has(timerKey)) {
      clearTimeout(typingTimers.get(timerKey));
    }
    // Відправляємо статус друкування
    socket.to(`conversation_${conversationId}`).emit('chat:typing', {
      userId,
      isTyping
    });
    // Автоматично знімаємо статус через 3 секунди
    if (isTyping) {
      const timer = setTimeout(() => {
        socket.to(`conversation_${conversationId}`).emit('chat:typing', {
          userId,
          isTyping: false
        });
        typingTimers.delete(timerKey);
      }, 3000);
      typingTimers.set(timerKey, timer);
    }
  });
  // Позначення як прочитане
  socket.on('chat:read', ({ conversationId }) => {
    const db = getDb();
    db.prepare(`
      UPDATE chat_messages
      SET is_read = 1
      WHERE conversation_id = ? AND sender_id != ?
    `).run(conversationId, userId);
    // Повідомляємо інших учасників
    socket.to(`conversation_${conversationId}`).emit('chat:read', {
      conversationId,
      userId
    });
  });
}
```

Header.jsx

```
import { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { motion, AnimatePresence } from 'framer-motion';
import { Menu, X, User, LogOut, Calendar, MessageCircle, Bell } from 'lucide-react';
import { useAuthStore } from '../../store/authStore';
import Avatar from '../ui/Avatar';
import Button from '../ui/Button';
function Header() {
  const [isMenuOpen, setIsMenuOpen] = useState(false);
  const [isUserMenuOpen, setIsUserMenuOpen] = useState(false);
  const { isAuthenticated, user, logout } = useAuthStore();
  const navigate = useNavigate();
  const handleLogout = async () => {
    await logout();
    navigate('/');
  };
  const getDashboardLink = () => {
    switch (user?.role) {
```

```
      case 'admin': return '/admin/dashboard';
      case 'master': return '/master/dashboard';
      default: return '/client/dashboard';
    }
  };
  const navLinks = [
    { to: '/catalog', label: 'Послуги' },
    { to: '/masters', label: 'Майстри' },
    { to: '/about', label: 'Про нас' },
    { to: '/contacts', label: 'Контакти' }
  ];
  return (
    <header className="fixed top-0 left-0 right-0 z-50 glass">
      <div className="container mx-auto px-4">
        <div className="flex items-center justify-between h-16 md:h-20">
          {/* Логотип */}
          <Link to="/" className="flex items-center gap-2">
            <span className="text-2xl md:text-3xl font-display text-
gradient">Шарм</span>
          </Link>
          {/* Desktop навігація */}
          <nav className="hidden md:flex items-center gap-8">
            {navLinks.map((link) => (
              <Link
                key={link.to}
                to={link.to}
                className="text-[#F5F5F0]/80 hover:text-gold transition-colors"
              >
                {link.label}
              </Link>
            ))}
          </nav>
          {/* Права частина */}
          <div className="flex items-center gap-4">
            {isAuthenticated ? (
              <div className="relative">
                <button
                  onClick={() => setIsUserMenuOpen(!isUserMenuOpen)}
                  className="flex items-center gap-2 hover:opacity-80 transition-
opacity"
                >
                  <Avatar
                    src={user?.avatar_url}
                    firstName={user?.first_name}
                    lastName={user?.last_name}
                    size="sm"
                  />
                  <span className="hidden md:block text-sm">{user?.first_name}</span>
                </button>
                <AnimatePresence>
                  {isUserMenuOpen && (
                    <motion.div
                      initial={{ opacity: 0, y: 10 }}
                      animate={{ opacity: 1, y: 0 }}
                      exit={{ opacity: 0, y: 10 }}
                      className="absolute right-0 mt-2 w-56 card p-2"
                    >
                      <div className="px-3 py-2 border-b border-gold/20 mb-2">
                        <p className="font-medium">{user?.first_name}
{user?.last_name}</p>
                        <p className="text-sm text-[#9A9A9A]">{user?.email}</p>
                      </div>
                      <Link
                        to={getDashboardLink()}
```

```
                        className="flex items-center gap-2 px-3 py-2 rounded-lg
hover:bg-gold/10 transition-colors"
                        onClick={() => setIsUserMenuOpen(false)}
                      >
                        <User size={18} />
                        <span>Особистий кабінет</span>
                      </Link>
                      {user?.role === 'client' && (
                        <>
                          <Link
                            to="/client/bookings"
                            className="flex items-center gap-2 px-3 py-2 rounded-lg
hover:bg-gold/10 transition-colors"
                            onClick={() => setIsUserMenuOpen(false)}
                          >
                            <Calendar size={18} />
                            <span>Мої записи</span>
                          </Link>
                          <Link
                            to="/client/chat"
                            className="flex items-center gap-2 px-3 py-2 rounded-lg
hover:bg-gold/10 transition-colors"
                            onClick={() => setIsUserMenuOpen(false)}
                          >
                            <MessageCircle size={18} />
                            <span>Чат</span>
                          </Link>
                        </>
                      )}
                      <button
                        onClick={handleLogout}
                        className="w-full flex items-center gap-2 px-3 py-2 rounded-lg
hover:bg-error/10 text-error transition-colors"
                      >
                        <LogOut size={18} />
                        <span>Вийти</span>
                      </button>
                    </motion.div>
                  )}
                </AnimatePresence>
              </div>
            ) : (
              <div className="hidden md:flex items-center gap-3">
                <Link to="/login">
                  <Button variant="ghost">Увійти</Button>
                </Link>
                <Link to="/register">
                  <Button>Реєстрація</Button>
                </Link>
              </div>
            )}
            {/* Mobile menu button */}
            <button
              onClick={() => setIsMenuOpen(!isMenuOpen)}
              className="md:hidden p-2 text-[#F5F5F0]"
            >
              {isMenuOpen ? <X size={24} /> : <Menu size={24} />}
            </button>
          </div>
        </div>
      </div>
      {/* Mobile menu */}
      <AnimatePresence>
        {isMenuOpen && (
          <motion.div
```

```
            initial={{ opacity: 0, height: 0 }}
            animate={{ opacity: 1, height: 'auto' }}
            exit={{ opacity: 0, height: 0 }}
            className="md:hidden border-t border-gold/20"
          >
            <nav className="container mx-auto px-4 py-4 space-y-2">
              {navLinks.map((link) => (
                <Link
                  key={link.to}
                  to={link.to}
                  className="block py-2 text-[#F5F5F0]/80 hover:text-gold"
                  onClick={() => setIsMenuOpen(false)}
                >
                  {link.label}
                </Link>
              ))}
              {!isAuthenticated && (
                <div className="pt-4 space-y-2">
                  <Link to="/login" onClick={() => setIsMenuOpen(false)}>
                    <Button variant="secondary" className="w-full">Увійти</Button>
                  </Link>
                  <Link to="/register" onClick={() => setIsMenuOpen(false)}>
                    <Button className="w-full">Реєстрація</Button>
                  </Link>
                </div>
              )}
            </nav>
          </motion.div>
        )}
      </AnimatePresence>
    </header>
  );
}
export default Header;
```

Footer.jsx

```
import { Link } from 'react-router-dom';
import { MapPin, Phone, Mail, Clock, Instagram, Facebook } from 'lucide-react';
function Footer() {
  return (
    <footer className="bg-dark-lighter border-t border-gold/20">
      <div className="container mx-auto px-4 py-12">
        <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-8">
          {/* Про салон */}
          <div>
            <h3 className="text-2xl font-display text-gradient mb-4">Шарм</h3>
            <p className="text-[#9A9A9A] mb-4">
              Професійний догляд за вашою красою. Досвідчені майстри, сучасні
технології, індивідуальний підхід.
            </p>
            <div className="flex gap-4">
              <a href="#" className="text-[#9A9A9A] hover:text-gold transition-
colors">
                <Instagram size={20} />
              </a>
              <a href="#" className="text-[#9A9A9A] hover:text-gold transition-
colors">
                <Facebook size={20} />
              </a>
            </div>
          </div>
          {/* Навігація */}
          <div>
```

```
            <h4 className="text-lg font-semibold mb-4">Навігація</h4>
            <ul className="space-y-2">
              <li>
                <Link to="/catalog" className="text-[#9A9A9A] hover:text-gold
transition-colors">
                  Послуги
                </Link>
              </li>
              <li>
                <Link to="/masters" className="text-[#9A9A9A] hover:text-gold
transition-colors">
                  Майстри
                </Link>
              </li>
              <li>
                <Link to="/about" className="text-[#9A9A9A] hover:text-gold
transition-colors">
                  Про нас
                </Link>
              </li>
              <li>
                <Link to="/contacts" className="text-[#9A9A9A] hover:text-gold
transition-colors">
                  Контакти
                </Link>
              </li>
            </ul>
          </div>
          {/* Контакти */}
          <div>
            <h4 className="text-lg font-semibold mb-4">Контакти</h4>
            <ul className="space-y-3">
              <li className="flex items-start gap-3 text-[#9A9A9A]">
                <MapPin size={18} className="text-gold mt-1 flex-shrink-0" />
                <span>вул. Івана Франка, 25, Львів</span>
              </li>
              <li className="flex items-center gap-3 text-[#9A9A9A]">
                <Phone size={18} className="text-gold flex-shrink-0" />
                <a href="tel:+380321234567" className="hover:text-gold transition-
colors">
                  +38 (032) 123-45-67
                </a>
              </li>
              <li className="flex items-center gap-3 text-[#9A9A9A]">
                <Mail size={18} className="text-gold flex-shrink-0" />
                <a href="mailto:info@sharm.ua" className="hover:text-gold transition-
colors">
                  info@sharm.ua
                </a>
              </li>
            </ul>
          </div>
          {/* Графік роботи */}
          <div>
            <h4 className="text-lg font-semibold mb-4">Графік роботи</h4>
            <ul className="space-y-2 text-[#9A9A9A]">
              <li className="flex items-center gap-3">
                <Clock size={18} className="text-gold" />
                <span>Пн–Пт: 09:00 – 20:00</span>
              </li>
              <li className="pl-8">Сб: 10:00 – 18:00</li>
              <li className="pl-8">Нд: вихідний</li>
            </ul>
          </div>
        </div>
```

```
        {/* Copyright */}
        <div className="mt-12 pt-8 border-t border-gold/20 text-center text-[#9A9A9A]
text-sm">
          <p>&copy; {new Date().getFullYear()} Салон краси «Шарм». Всі права
захищені.</p>
        </div>
      </div>
    </footer>
  );
}
export default Footer;
```

Avatar.jsx

```
import * as AvatarPrimitive from '@radix-ui/react-avatar';
import { cn, getInitials } from '../../lib/utils';
function Avatar({ src, alt, firstName, lastName, size = 'md', className }) {
  const sizes = {
    xs: 'w-6 h-6 text-[10px]',
    sm: 'w-8 h-8 text-xs',
    md: 'w-10 h-10 text-sm',
    lg: 'w-12 h-12 text-base',
    xl: 'w-16 h-16 text-lg',
    '2xl': 'w-24 h-24 text-2xl'
  };
  return (
    <AvatarPrimitive.Root
      className={cn(
        'relative flex shrink-0 overflow-hidden rounded-full',
        sizes[size],
        className
      )}
    >
      <AvatarPrimitive.Image
        src={src}
        alt={alt || `${firstName} ${lastName}`}
        className="aspect-square h-full w-full object-cover"
      />
      <AvatarPrimitive.Fallback
        className="flex h-full w-full items-center justify-center rounded-full bg-
gold/20 text-gold font-medium"
      >
        {getInitials(firstName, lastName)}
      </AvatarPrimitive.Fallback>
    </AvatarPrimitive.Root>
  );
}
export default Avatar;
```

Button.jsx

```
import { forwardRef } from 'react';
import { cn } from '../../lib/utils';
import { Loader2 } from 'lucide-react';
const variants = {
  primary: 'btn-primary',
  secondary: 'btn-secondary',
  ghost: 'btn-ghost',
  danger: 'px-6 py-3 bg-error text-white rounded-lg hover:bg-error/90 transition-all
duration-300'
};
const sizes = {
  sm: 'px-4 py-2 text-sm',
```

```
  md: 'px-6 py-3',
  lg: 'px-8 py-4 text-lg'
};
const Button = forwardRef(({
  className,
  variant = 'primary',
  size = 'md',
  isLoading = false,
  disabled,
  children,
  ...props
}, ref) => {
  return (
    <button
      ref={ref}
      className={cn(
        variants[variant],
        sizes[size],
        'inline-flex items-center justify-center gap-2',
        className
      )}
      disabled={disabled || isLoading}
      {...props}
    >
      {isLoading && <Loader2 className="w-4 h-4 animate-spin" />}
      {children}
    </button>
  );
});
Button.displayName = 'Button';
export default Button;
```

Card.jsx

```
import { cn } from '../../lib/utils';
function Card({ className, hover = false, children, ...props }) {
  return (
    <div
      className={cn(
        'card',
        hover && 'card-hover',
        className
      )}
      {...props}
    >
      {children}
    </div>
  );
}
function CardHeader({ className, children, ...props }) {
  return (
    <div className={cn('mb-4', className)} {...props}>
      {children}
    </div>
  );
}
function CardTitle({ className, children, ...props }) {
  return (
    <h3 className={cn('text-h4 text-[#F5F5F0]', className)} {...props}>
      {children}
    </h3>
  );
}
function CardDescription({ className, children, ...props }) {
```

```
  return (
    <p className={cn('text-sm text-[#9A9A9A]', className)} {...props}>
      {children}
    </p>
  );
}
function CardContent({ className, children, ...props }) {
  return (
    <div className={cn('', className)} {...props}>
      {children}
    </div>
  );
}
function CardFooter({ className, children, ...props }) {
  return (
    <div className={cn('mt-4 pt-4 border-t border-gold/20', className)} {...props}>
      {children}
    </div>
  );
}
export { Card, CardHeader, CardTitle, CardDescription, CardContent, CardFooter };
```

DatePicker.jsx

```
import { useState, useRef, useEffect } from 'react';
import { createPortal } from 'react-dom';
import { ChevronLeft, ChevronRight, Calendar } from 'lucide-react';
import { cn } from '../../lib/utils';
// Українська локалізація
const ukMonths = [
  'Січень', 'Лютий', 'Березень', 'Квітень', 'Травень', 'Червень',
  'Липень', 'Серпень', 'Вересень', 'Жовтень', 'Листопад', 'Грудень'
];
const ukDays = ['Пн', 'Вт', 'Ср', 'Чт', 'Пт', 'Сб', 'Нд'];
function DatePicker({ value, onChange, placeholder = 'Оберіть дату', className,
minDate, maxDate }) {
  const [isOpen, setIsOpen] = useState(false);
  const [position, setPosition] = useState({ top: 0, left: 0 });
  const [viewDate, setViewDate] = useState(() => {
    if (value) return new Date(value);
    return new Date();
  });
  const buttonRef = useRef(null);
  const calendarRef = useRef(null);
  // Позиціонування та закриття
  useEffect(() => {
    if (isOpen && buttonRef.current) {
      const rect = buttonRef.current.getBoundingClientRect();
      setPosition({
        top: rect.bottom + window.scrollY + 8,
        left: rect.left + window.scrollX
      });
    }
  }, [isOpen]);
  // Закриття при кліку поза компонентом
  useEffect(() => {
    const handleClickOutside = (e) => {
      if (
        buttonRef.current && !buttonRef.current.contains(e.target) &&
        calendarRef.current && !calendarRef.current.contains(e.target)
      ) {
        setIsOpen(false);
      }
    };
```

```
    document.addEventListener('mousedown', handleClickOutside);
    return () => document.removeEventListener('mousedown', handleClickOutside);
  }, []);
  // Закриття при скролі
  useEffect(() => {
    const handleScroll = () => setIsOpen(false);
    if (isOpen) {
      window.addEventListener('scroll', handleScroll, true);
      return () => window.removeEventListener('scroll', handleScroll, true);
    }
  }, [isOpen]);
  const year = viewDate.getFullYear();
  const month = viewDate.getMonth();
  // Генерація днів місяця
  const generateDays = () => {
    const firstDay = new Date(year, month, 1);
    const lastDay = new Date(year, month + 1, 0);
    const daysInMonth = lastDay.getDate();
    let startDay = firstDay.getDay() - 1;
    if (startDay < 0) startDay = 6;
    const days = [];
    for (let i = 0; i < startDay; i++) {
      days.push(null);
    }
    for (let i = 1; i <= daysInMonth; i++) {
      days.push(i);
    }
    return days;
  };
  const handlePrevMonth = () => {
    setViewDate(new Date(year, month - 1, 1));
  };
  const handleNextMonth = () => {
    setViewDate(new Date(year, month + 1, 1));
  };
  const handleSelectDay = (day) => {
    if (!day) return;
    const selectedDate = new Date(year, month, day);
    const dateStr = selectedDate.toISOString().split('T')[0];
    if (minDate && dateStr < minDate) return;
    if (maxDate && dateStr > maxDate) return;
    onChange(dateStr);
    setIsOpen(false);
  };
  const isSelected = (day) => {
    if (!day || !value) return false;
    const dateStr = new Date(year, month, day).toISOString().split('T')[0];
    return dateStr === value;
  };
  const isDisabled = (day) => {
    if (!day) return true;
    const dateStr = new Date(year, month, day).toISOString().split('T')[0];
    if (minDate && dateStr < minDate) return true;
    if (maxDate && dateStr > maxDate) return true;
    return false;
  };
  const isToday = (day) => {
    if (!day) return false;
    const today = new Date();
    return day === today.getDate() && month === today.getMonth() && year ===
today.getFullYear();
  };
  const formatDisplayDate = () => {
    if (!value) return placeholder;
    const d = new Date(value);
```

```
    return `${d.getDate()} ${ukMonths[d.getMonth()].toLowerCase()}
${d.getFullYear()}`;
  };
  const days = generateDays();
  const calendarContent = (
    <div
      ref={calendarRef}
      style={{ position: 'absolute', top: position.top, left: position.left }}
      className="z-[99999] p-4 bg-dark-lighter border border-gold/20 rounded-xl
shadow-2xl min-w-[280px]"
    >
      {/* Заголовок */}
      <div className="flex items-center justify-between mb-4">
        <button
          type="button"
          onClick={handlePrevMonth}
          className="p-1 hover:bg-gold/10 rounded-lg transition-colors"
        >
          <ChevronLeft size={20} />
        </button>
        <span className="font-display">
          {ukMonths[month]} {year}
        </span>
        <button
          type="button"
          onClick={handleNextMonth}
          className="p-1 hover:bg-gold/10 rounded-lg transition-colors"
        >
          <ChevronRight size={20} />
        </button>
      </div>
      {/* Дні тижня */}
      <div className="grid grid-cols-7 gap-1 mb-2">
        {ukDays.map((day) => (
          <div key={day} className="text-center text-xs text-[#9A9A9A] py-1">
            {day}
          </div>
        ))}
      </div>
      {/* Дні місяця */}
      <div className="grid grid-cols-7 gap-1">
        {days.map((day, index) => (
          <button
            key={index}
            type="button"
            onClick={() => handleSelectDay(day)}
            disabled={isDisabled(day)}
            className={cn(
              'w-9 h-9 rounded-lg text-sm transition-colors',
              day === null && 'invisible',
              isSelected(day) && 'bg-gold text-dark font-medium',
              !isSelected(day) && !isDisabled(day) && 'hover:bg-gold/20',
              isToday(day) && !isSelected(day) && 'border border-gold/50',
              isDisabled(day) && day !== null && 'text-[#9A9A9A]/30 cursor-not-
allowed'
            )}
          >
            {day}
          </button>
        ))}
      </div>
      {/* Кнопка "Сьогодні" */}
      <button
        type="button"
        onClick={() => {
```

```
            const today = new Date().toISOString().split('T')[0];
            if ((!minDate || today >= minDate) && (!maxDate || today <= maxDate)) {
              onChange(today);
              setIsOpen(false);
            }
          }}
          className="w-full mt-3 py-2 text-sm text-gold hover:bg-gold/10 rounded-lg
transition-colors"
        >
          Сьогодні
        </button>
      </div>
    );
    return (
      <div className="relative">
        {/* Кнопка */}
        <button
          ref={buttonRef}
          type="button"
          onClick={() => setIsOpen(!isOpen)}
          className={cn(
            'input w-full text-left flex items-center justify-between gap-2',
            !value && 'text-[#9A9A9A]',
            className
          )}
        >
          <span>{formatDisplayDate()}</span>
          <Calendar size={18} className="text-[#9A9A9A]" />
        </button>

        {/* Календар через портал */}
        {isOpen && createPortal(calendarContent, document.body)}
      </div>
    );
}
export default DatePicker;
```