ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)

«___» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
РОЗРОБКА ТА ВПРОВАДЖЕННЯ ВЕБ-САЙТУ ДЛЯ ЗАМОВЛЕНЬ ПОСЛУГ РЕМОНТУ ТА ПОШИВУ ОДЯГУ
зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавра

Виконавець роботи Халявка Гаврило Валерійович
_____ «____»_____ 2025 р.
(підпис)

Науковий керівник к.пед.н., доцент, Кошова Оксана Петрівна
_____ «____»_____ 2025 р.
(підпис)

Рецензент

ПОЛТАВА 2025 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Олена ОЛЬХОВСЬКА
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ І КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Розробка сайту з ремонту та пошиву одягу**»**
зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Халявка Гаврило Валерійович
Затверджена наказом ректора № 177-Н від «10» вересня 2024 р.
Термін подання студентом роботи «___»_____ 2025 р.
Вихідні дані до кваліфікаційної роботи: статті та документації з теми розробки сайтів.

Зміст пояснювальної записки (перелік питань, які потрібно розробити)
ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
1.1. Мета та задачі дослідження
1.2. Визначення вимог до функціоналу сайту з ремонту та пошиття одягу
РОЗДІЛ 2. ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ
2.1. Аналіз наявних платформ і засобів для створення веб-сайтів
2.2. Порівняльний аналіз популярних рішень для створення сайтів
2.3. Переваги та недоліки існуючих сайтів з ремонту та пошиття одягу
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Обґрунтування вибору технологій та інструментів
3.2. Схема реалізації та логіка роботи сайту
3.3. Структура та компоненти веб-застосунку
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1. Етапи реалізації функціоналу веб-застосунку
4.2. Реалізований функціонал сайту для користувачів
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
Перелік графічного матеріалу: 1 аркуш блок-схем, 33 ілюстрації.

Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Кошова О.П.		
Інформаційний огляд	Кошова О.П.		
Теоретична частина	Кошова О.П.		
Практична частина	Кошова О.П.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
Вступ		
Вивчення методичних рекомендацій та стандартів та звіт керівнику		
Постановка задачі		
Інформаційний огляд джерел бібліотек та інтернету		
Теоретична частина		
Практична частина		
Закінчення оформлення		
Доповідь студента на кафедрі		
Доробка (за необхідністю), рецензування		

Дата видачі завдання «__» _____2024 р.

Здобувач вищої освіти _____ Халявка Гаврило Валерійович
(підпис)
Науковий керівник _____ к.пед.н., доц. Кошова О.П.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Результати захисту кваліфікаційної роботи
Кваліфікаційна робота оцінена на _____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 2025 р.
Секретар ЕК_____ _____
(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 2024 р.

Погоджено
Науковий керівник _____
к.пед.н., Оксана КОШОВА
«____» _____ 2024 р.

План
дипломного проекту з фаху
спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
на тему «Розробка сайту з ремонту та пошиття одягу»
Прізвище, ім'я, по батькові Халявка Гаврило Валерійович

ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
1.1. Мета та задачі дослідження
1.2. Визначення вимог до функціоналу сайту з ремонту та пошиття одягу
РОЗДІЛ 2. ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ
2.1. Аналіз наявних платформ і засобів для створення веб-сайтів
2.2. Порівняльний аналіз популярних рішень для створення сайтів
2.3. Переваги та недоліки існуючих сайтів з ремонту та пошиття одягу
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Обґрунтування вибору технологій та інструментів
3.2. Схема реалізації та логіка роботи сайту
3.3. Структура та компоненти веб-застосунку
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1. Етапи реалізації функціоналу веб-застосунку
4.2. Реалізований функціонал сайту для користувачів
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

Здобувач вищої освіти _____ Г. В. Халявка
(підпис)
«____» _____ 2024 р.

РЕФЕРАТ

Записка: 155 сторінок, 33 рисунока, 1 таблиця, 1 блок-схема, 13 літературних джерел.

Laravel, Веброзробка, База даних, Аутентифікація,Інтерфейс користувача

Об'єкт розробки – вебсайт для надання послуг з ремонту та пошиву одягу.

Мета роботи розробити сучасний, адаптивний вебсайт для надання послуг з ремонту та пошиву одягу з використанням фреймворку Laravel та Tailwind CSS.

Методи дослідження – методи аналізу вимог користувача, проєктування архітектури вебсайту, розробка з використанням фреймворку Laravel та утилітного CSS-фреймворку Tailwind CSS. Для зберігання даних застосовано СУБД MySQL. Застосовано адаптивну верстку для забезпечення комфортної роботи сайту на різних пристроях.

У роботі визначено ключові вимоги до функціоналу сайту; проведено аналіз сучасних вебтехнологій, придатних для створення сервісів із надання побутових послуг; виконано порівняння з існуючими аналогами сайтів у цій сфері.

Описано вибір технологій, методологію розробки, побудовано блок-схему логіки роботи сайту, створено архітектуру бази даних і реалізовано інтерфейс користувача. Особливу увагу приділено функціоналу онлайн-замовлення послуг, управління контентом та зворотному зв'язку з користувачем.

Сайт протестовано на відповідність технічним та функціональним вимогам, створено інструкцію з користування, а також додано графічні матеріали: блок-схему, діаграму класів, макети інтерфейсу тощо.

За темою дипломної роботи було підготовлено тезу для участі в науково-практичній конференції.

Зміст

ВСТУП

У сучасному світі інтернет став невід'ємною частиною життя більшості людей. Онлайн-присутність для бізнесу вже давно не є розкішшю, а обов'язковим елементом успішної діяльності. Сфера побутових послуг, зокрема ремонт та пошив одягу, не є винятком. Замовники дедалі частіше шукають майстрів саме через інтернет, очікуючи зручності, швидкого зворотного зв’язку та можливості онлайн-замовлення.

Незважаючи на активний розвиток вебтехнологій, багато майстерень з ремонту одягу досі не мають власного сайту або використовують застарілі рішення, що не забезпечують належного користувацького досвіду. Це створює попит на сучасні, адаптивні вебсайти з простим інтерфейсом, зручним адмініструванням та функцією онлайн-бронювання послуг.

Актуальність теми зумовлена необхідністю цифровізації малого бізнесу, підвищення ефективності взаємодії між клієнтами та виконавцями послуг, а також популярністю фреймворків, які дозволяють створювати якісні вебрішення з мінімальними витратами часу та ресурсів.

Мета даної бакалаврської роботи – розробити сучасний, адаптивний вебсайт для надання послуг з ремонту та пошиву одягу з використанням фреймворку Laravel та Tailwind CSS.

У межах дослідження буде розглянуто існуючі рішення, обрано технології, описано процес проєктування, реалізації та тестування програмного забезпечення. Результатом стане функціональний сайт, що дозволяє клієнтам ознайомлюватись з переліком послуг, замовляти ремонт чи пошив одягу, а адміністраторам – ефективно керувати контентом і заявками.

1 ПОСТАНОВКА ЗАДАЧІ

1.1. Мета та задачі дослідження

Мета – розробка функціонального, зручного та ефективного веб-ресурсу для ательє, використовуючи сучасні технології веб-розробки, такі як **Laravel** для створення backend-частини, **Tailwind CSS** для стилізації інтерфейсу та **MySQL** для управління базою даних. Важливим аспектом є розробка інтуїтивно зрозумілого дизайну та реалізація механізмів взаємодії з клієнтами для підвищення якості обслуговування.

Для досягнення поставленої мети передбачається виконання наступних завдань:

Аналіз наявних рішень – дослідження сучасних веб-сайтів для майстерень з ремонту та пошиття одягу, аналіз їх функціоналу та зручності використання.

Проектування архітектури сайту – розробка логічної структури ресурсу, визначення необхідного набору сторінок, функцій та інтерактивних елементів.

Створення backend-частини – реалізація серверної логіки та обробки запитів за допомогою **Laravel**, налаштування зв’язку між системними компонентами.

Розробка UI/UX дизайну – впровадження адаптивного інтерфейсу з використанням **Tailwind CSS**, забезпечення зручної навігації для користувачів.

Інтеграція бази даних – організація ефективного збереження та управління даними за допомогою **MySQL**, оптимізація запитів для швидкої обробки інформації.

Реалізація функціоналу – розробка можливостей реєстрації користувачів, прийому замовлень, перегляду статусу виконання робіт.

Таким чином, задачі спрямовані на створення ефективного рішення, яке не лише автоматизує процеси ательє, а й підвищить рівень клієнтського обслуговування, дозволяючи майстру зосередитися на виконанні своїх основних завдань. Грамотна розробка архітектури, дизайн та функціонал дозволять досягти поставленої мети та створити сучасний ресурс, який відповідатиме вимогам як власника ательє, так і йогоІ клієнтів.

1.2. Визначення вимог до функціоналу сайту з ремонту та пошиття одягу

Розробка сайту для ремонту та пошиття одягу вимагає чіткого визначення функціональних вимог, які забезпечать його ефективну роботу, зручність користування та відповідність потребам майстра і клієнта. Функціональні вимоги можна поділити на **основні** та **технічні**.

Основні вимоги:

Реєстрація та авторизація користувачів – можливість створення облікового запису для клієнтів та майстрів ательє, авторизація через електронну пошту або соціальні мережі.

Форма замовлення – інтуїтивно зрозумілий механізм створення замовлення, вибору типу ремонту чи пошиття, вказання параметрів, прикріплення фото та опису.

Статус виконання замовлення – можливість перегляду поточного стану роботи, етапів виконання замовлення та сповіщення про готовність.

Статус оплати – можливість огляду суми платежу онлайн, виставлення рахунку.

Технічні вимоги:

Безпека даних – реалізація захисту персональних даних користувачів, шифрування інформації та безпечне зберігання паролів.

Мобільна адаптивність – повноцінна робота сайту на смартфонах та планшетах завдяки адаптивному дизайну.

Оптимізація швидкості роботи – швидке завантаження сторінок та оптимізація запитів до бази даних **MySQL**.

Адміністративна панель – зручний інтерфейс для управління замовленнями, користувачами, послугами та фінансами.

2 ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ

2.1. Аналіз наявних платформ і засобів для створення веб-сайтів

Для майстерні з ремонту та пошиву одягу потрібен простий веб-сайт з інформацією про послуги та можливістю зв’язку з клієнтами (наприклад, через форми заявок). Розглянемо кілька безкоштовних платформ для створення таких сайтів (WordPress.com, Wix Free, Google Sites, Tilda Free, GitHub Pages з Jekyll) та порівняємо їх із підходом “самописного” сайту на основі Laravel + Tailwind CSS. Оцінюватимемо за критеріями простоти використання, кастомізації дизайну/функціоналу, наявності форм (чи бази даних) для збору замовлень та адаптивності (відображення на різних пристроях).

WordPress.com (Free Plan) (Рис 2.1). Хмарна CMS, що пропонує баланс між простотою та функціональністю. Інтерфейс адміністратора відносно інтуїтивний (блочний редактор), тому навіть непрофесіонали можуть створювати сайтиwoblogger.com. Безкоштовний план включає Jetpack (базові SEO-інструменти, статистику, соціальні віджети)wordpress.com, 1 ГБ хостингу та декілька привабливих тем. Однак кастомізація обмежена: можна вибирати лише з безкоштовних вбудованих тем та налаштовувати стандартні опції (кольори, шрифти)thimpress.comthimpress.com. Підтримки власних плагінів або повної стилізації CSS без підписки немає. Власної бази даних у WordPress.com звичайно використовується за лаштунками CMS, але для складніших форм потрібні плагіни (які на Free-плані відсутні). Тобто є лише прості вбудовані контактні блоки (через Jetpack), без власного модуля замовлень. Сучасні теми WordPress зазвичай адаптивні, тому сайт коректно відображається на комп’ютері, планшеті чи смартфоні.

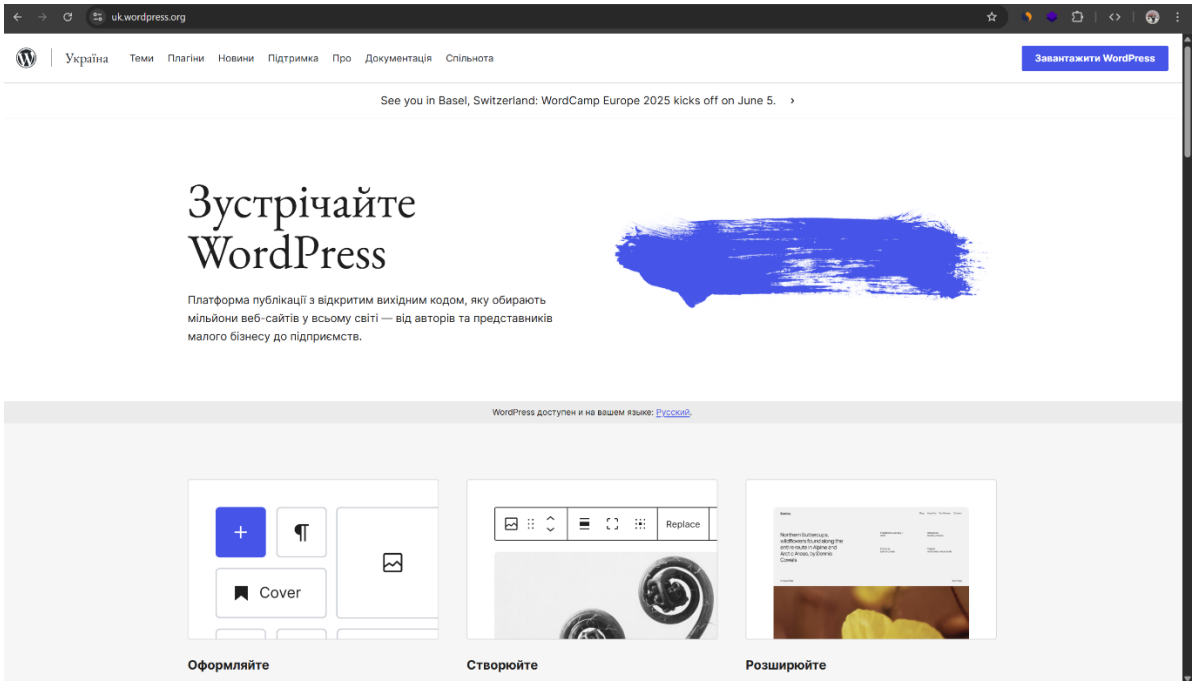

Рисунок 2.1 Головна сторінка WordPress.com

Wix (Free Plan). Конструктор сайтів з дуже зручним візуальним інтерфейсом. Редактор Wix працює за принципом drag-&-drop, тому можна буквально перетягувати блоки на сторінку й миттєво бачити результатbluehost.com. Платформа пропонує десятки готових шаблонів та секцій (галереї, тексти, кнопки тощо)bluehost.com. Для непрофесіоналів це великий плюс: не треба писати код, достатньо настроїти шаблон. Крім того, у Wix є маркет додатків (App Market) із безкоштовними інструментами: можна додати контактні форми, інтегрувати соцмережі, покращити SEO та інші функції безкоштовноbluehost.com. Безкоштовний план дає 500 МБ дискового простору і 1 ГБ трафіку, але накладає обмеження: на сайті відображатимуться банери Wix і адреса буде формату sitename.wixsite.com/...bluehost.com. Налаштування дизайну обмежене вибором шаблону (після початку роботи його складно змінити). Також у Wix немає власної бази даних – всі дані статичні, а форми/заявки збираються через додатки (наприклад, прості контактні форми)bluehost.com. Однак усі шаблони Wix автоматично оптимізовані під мобільні пристроїbluehost.com, тож сайт виглядає гарно на смартфоні і планшеті.

Google Sites. Абсолютно безкоштовний та простий конструктор від Google. Не потребує кодування – все робиться перетягуванням блоків (drag-&-

drop)sites.google.com. Google Sites інтегрується з інструментами Google (Forms, Sheets, Docs), тому збір заявок зазвичай реалізують через вбудовані Google-Форми. Функціонал дизайну дуже обмежений: лише кілька простих шаблонів та базові типи блоків (текст, зображення, відео). Власної бази даних немає – дані форм можна записувати в Google Sheets. Перевага – це гарантія адаптивності і надійності Google: всі сайти на Google Sites автоматично адаптуються під будь-який розмір екранаsites.google.com. Набір можливостей невеликий, але для простого інформаційного сайту це може бути достатньо.

Laravel + Tailwind CSS. Повноцінна розробка "з нуля" на базі PHP-фреймворка Laravel (який безкоштовний). Потрібно самостійно придбати хостинг (можливі платні або безкоштовні варіанти) і реалізувати все вручну. Перевага – максимальна кастомізація: ви отримуєте повний контроль над будь-яким дизайном і функціоналом. Tailwind CSS дозволяє легко робити адаптивну верстку, бо кожному класу можна задавати умови для різних розмірів екрана, що суттєво спрощує реалізацію адаптивностіtailwindcss.com. Laravel має потужну підтримку баз даних (Eloquent ORM), тому форми замовлень та обробка даних реалізуються на рівні серверу повноцінно. Але геть очевидний недолік – потрібні розробники з досвідом. Це значно складніше й дорожче в реалізації і підтримціwoblogger.com. Такий підхід варто обирати, якщо потрібен унікальний функціонал або складна логіка, яку неможливо реалізувати в готових конструкторах.

2.2. Порівняльний аналіз популярних рішень для створення сайтів

WordPress.com (безкоштовний план)

Сторінки та контент: WordPress.com дозволяє створювати базові сторінки та записи з текстом і зображеннями (наприклад, «Про нас», опис послуг, галерею фото). Однак можливості налаштування дизайну обмежені стандартними темами, а додаткова функціональність реалізується плагінами.

Форма зворотного зв’язку: У безкоштовному плані немає вбудованого конструктора контактної форми – її можна додати тільки через плагін (доступний починаючи з бізнес-плану)wpbeginner.com. Інакше доводиться вбудовувати зовнішні форми (наприклад, Google Forms).

Карта: Немає готового блоку карти – карту Google можна вставити вручну через HTML-iframe.

Стандартні функції: Є набір безкоштовних тем базовий редактор блоків і Jetpack для аналітики. Для SEO та інших просунутих опцій потрібні плагіни (що недоступні на безкоштовному плані)wpbeginner.com.

Обмеження безкоштовного плану: Сайт працюватиме на субдомені *.wordpress.com із рекламою WordPress.com, виділено обмежений обсяг хостингу (кілька ГБ). Немає можливості встановити сторонні плагіни чи завантажити власні темиwpbeginner.comwpbeginner.com, а також редагувати CSS.

Підходить для початківців та некомерційних проектів, де треба тільки представити інформацію й контакти, без складних інтерактивних елементів. Добре підходить для блогів чи простих презентаційних сайтів малого бізнесу з бюджетом $0, але не для тих, хто хоче унікальний дизайн чи серйозну функціональність.

Wix (безкоштовний план)

Сторінки та контент: Конструктор «drag and drop» із великою бібліотекою адаптивних шаблонів (є варіанти для кравців/ремонтників). Легко створювати сторінки «Послуги», «Галерея» (фото), «Контакти» тощо та заповнювати їх.

Форма зворотного зв’язку: У редакторі Wix є вбудовані елементи контактних форм та безкоштовні додатки з App Market. Можна додати просту форму для заявок з опціями полів (наприклад, через Contact Form by POWR) безкоштовно (з обмеженням, наприклад, ~25 відправок на місяць).

Карта: Google Map легко додається як елемент через меню «Add→Contact & Forms→Maps»support.wix.com.

Стандартні функції: Автоматичне мобільне відображення, SSL, власний конструктор SEO (базовий), безкоштовна бібліотека медіа. Є доступ до сотень додатків (форми, соцмережі, SEO) в App Market.

Обмеження безкоштовного плану: Безкоштовний обсяг диску – 500 МБ, трафік – 1 ГБ. Сайт показуватиме банер із рекламою Wix і працюватиме лише на субдомені . Немає можливості підключити власний домен чи Google Analytics, а також відсутній функціонал eCommerce.

Підходить для: Нечутливих до витрат користувачів та початківців, яким потрібен швидкий дизайнерський сайт без кодування. Добре підходить малому бізнесу, фрілансерам, які можуть обійтися брендингом Wix і не потребують складних розширень. Не підходить тим, хто вимагає відсутності реклами або власного домену.

Google Sites

Сторінки та контент: Дуже простий конструктор – дозволяє створювати базові сторінки з текстом, зображеннями, навігацією. Підходить для розділів «Про майстерню», «Послуги», «Фотогалерея» тощо, але шаблони шаблонів обмежені.

Форма зворотного зв’язку: Форми реалізуються через Google Forms – їх можна вбудувати як елемент на сторінку. Нативного «конструктора заявки» немає.

Карта: Можна вставити Google Map через вбудоване меню «Вставити» → «Карта». Це стандартна функція конструктора.

Стандартні функції: Міцно інтегрований з екосистемою Google: напр., легко вбудовувати Google Docs, Calendar, Forms, Maps без додаткових налаштуваньsheetifycrm.comvendasta.com. Є базовий набір віджетів та адаптивний дизайн.

Обмеження безкоштовної версії: Без власного домену – сайти публікуються на адресах Google (щоб використовувати персональний домен, потрібно платити за Workspace)vendasta.com. Бібліотека шаблонів невелика, можливості дизайну дуже обмежені (мається лише стандартний набір шрифтів і блоків)vendasta.com. Загальна

пам’ять проекту обмежена ~15 ГБ у хмарі Google (надлишок потребуватиме платного розширення)vendasta.com. Немає вбудованих SEO-інструментів, блогів чи eCommerce.
Підходить для: Абсолютних новачків або невеликих компаній, які хочуть безкоштовно отримати простий сайт і вже використовують інші Google-сервіси. Зручно, якщо потрібна максимальна простота й інтеграція з Gmail/Drive. Однак низька гнучкість і складніші обмеження роблять Google Sites менш привабливими для бізнесів, що прагнуть оригінальний дизайн або розширену функціональність.

Laravel + Tailwind CSS

Сторінки та контент: Це не готова платформа, а власна розробка. Кожна сторінка і розділ (послуги, фотогалерея, контакти) створюються вручну через маршрути та шаблони Laravel. Для дизайну використовуються компоненти Tailwind CSS. Такий підхід дає необмежену гнучкість, але вимагає програмування.

Форма зворотного зв’язку: Повністю кастомізована – створюєте контролер та форму у шаблоні, обробляєте запит на сервері. Можна зберігати заявки в базі, відправляти на email без додаткових обмежень.

Карта: Інтеграція карти здійснюється вручну, наприклад через API Google Maps або вставку iframe у Blade-шаблон.
Стандартні функції: Laravel надає маршрутизацію, захист від атак, роботу з базою даних, логіку, а Tailwind полегшує стилізацію. Немає нічого «з коробки» для сайту майстерні – все треба налаштовувати самому.

Безкоштовний варіант: Laravel і Tailwind безкоштовні (open-source), але потрібно власне середовище (сервер/хостинг) для розміщення сайту. Зазвичай використовується платний хостинг або локальні сервери (наприклад, Laravel Sail), які коштують грошей.

Підходить для: Досвідчених розробників і компаній з нестандартними вимогами. Це підхід для тих, кому потрібна повна свобода – наприклад, складні інтеграції, власна логіка, маштабованість. Не підходить для непрофесіоналів або для швидкого старту: тут потрібен час на розробку і навички програмування

Платформа	Простота використання	Кастомізація (дизайн/функці)	База даних / форми	Адаптивність
WordPress.com	Порівняно проста CMS: інтуїтивна адмінпанель, блочний редактор	Вбудовані теми та налаштування кольорів; тільки безкоштовні теми; **немає** сторонніх плагінів	Контент зберігається в CMS; прості контактні форми через Jetpack (власної БД немає)	Теми зазвичай адаптивні
Wix (Free)	Дуже простий drag'n'drop редактор: підходить для непрофесіоналів	Десятки шаблонів; безкоштовні додатки (форми, віджети соцмереж, SEO)	Немає власної БД; форми/заявки збираються через додатки (контактні форми)	Шаблони автоматично оптимізовані під мобільні
Google Sites	Дуже простий конструктор Google: drag'n'drop, не потребує кодування	Мало налаштувань: кілька простих шаблонів і блоків (текст, зображення тощо)	Немає власної БД; дані форм записуються в Google Forms/Sheets	Автоматично адаптується на всіх екранах
Laravel + Tailwind	Вимагає знань з програмування PHP/Laravel	Максимальна кастомізація (власний код, довільний функціонал)	Повноцінна БД (Eloquent); форми та обробка даних за вашою логікою	Tailwind має зручні утиліти для адаптивності

Таблиця 1

2.3. Переваги та недоліки існуючих сайтів з ремонту та пошиття одягу

Сайт дизайн студії «MARYNAD»(https://marynad.com.ua/)

Плюси:

Користувачі можуть легко зрозуміти, які послуги пропонує студія.

Перегляд моделей допомагає користувачам краще зрозуміти стиль та якість роботи студії.

Присутні контактні дані та розташування на Google Maps, що полегшує зв'язок з клієнтами.

Мінуси:

Користувачі не можуть створювати облікові записи для замовлень або отримання додаткової інформації безпосередньо через сайт.

Відсутність додаткових функцій, таких як онлайн-замовлення або відстеження статусу замовлень.

Сайт проекту «TOMILANA» (https://tomilana.com/)

Плюси:

Забезпечує різноманітні функції, що робить використання сервісу зручним для користувачів.

Можливість реєстрації та авторизації через телефон та e-mail забезпечує зручність та гнучкість.

Присутність контактів та розташування на Google Maps полегшує пошук і зв’язок з ательє.

Можливість записатися онлайн безпосередньо через сайт економить час користувачів та підвищує зручність.

Мінуси:

Інтеграція всіх цих функцій може бути технічно складною і потребувати значних ресурсів.

Потребує постійної підтримки та оновлень для забезпечення стабільної роботи всіх функцій.

Безпека даних: Зберігання та обробка особистих даних користувачів потребує високого рівня безпеки та захисту від кібератак.

Сторінка ательє в соціальній мережі Instagram «porokh_atelier»

(https://www.instagram.com/porokh_atelier/):

Плюси:

На сторінці часто публікуються фотографії готових виробів, стадії роботи над замовленнями, а також короткі відео про процес пошиву.

Також тут можна знайти інформацію про послуги, ціни та контактні дані ательє.

Активна взаємодія з користувачем, відповідають на питання та приймають замовлення.

Мінуси:

Обмежена інформація: Instagram є візуальною платформою, тому текстова інформація може бути обмеженою. Можливо, важливі деталі про послуги або ціни не завжди будуть доступні.

Вміст Instagram складається з постів і сторіс, що можуть не забезпечувати зручної структури для пошуку конкретної інформації.

3 ТЕОРЕТИЧНА ЧАСТИНА

3.1. Опис обраних технологій

Для розробки сайту було обрано такі технології:

1. Visual Studio Code (VS Code) є безкоштовним, кросплатформним інтегрованим середовищем розробки (IDE), створеним компанією Microsoft. Воно характеризується високою продуктивністю, широкими можливостями розширення та зручністю у використанні для фахівців у галузі програмування. Завдяки своїй функціональності VS Code є оптимальним інструментом для створення веб-додатків, зокрема тих, що розробляються з використанням фреймворку Laravel.

Середовище розробки забезпечує базову підтримку таких мов програмування, як HTML, CSS, JavaScript і PHP, а також дозволяє встановлювати додаткові розширення для роботи з іншими мовами. У контексті розробки проєктів на основі Laravel важливу роль відіграє підтримка таких технологій, як PHP, шаблонізатор Blade, JavaScript, формат JSON та мова запитів SQL.

Однією з ключових функцій VS Code є тісна інтеграція з системою контролю версій Git. Ця функціональність забезпечує можливість відстеження змін у коді, створення комітів і гілок, а також синхронізацію з віддаленим репозиторієм GitHub без потреби у використанні зовнішнього клієнтського програмного забезпечення. Така інтеграція суттєво спрощує командну розробку та полегшує контроль версій.

Завдяки системі розширень VS Code може бути адаптований до повноцінної роботи з Laravel-проєктами. Зокрема, розширення для Blade забезпечують підсвічування синтаксису та автозаповнення інструкцій шаблонізатора, що сприяє зменшенню кількості помилок і пришвидшує написання шаблонного коду. Розширення PHP Intelephense пропонує інтелектуальні підказки, перевірку синтаксису та діагностику помилок у коді PHP. Плагін Laravel Artisan дозволяє виконувати команди Artisan безпосередньо з середовища редактора, а розширення Laravel Extra Intellisense підвищує якість підказок для маршрутів, фасадів і конфігурацій Laravel.

VS Code також має вбудовану підтримку інструмента GitHub Copilot — штучного інтелекту, який автоматично генерує фрагменти коду на основі контексту. Це дозволяє значно зменшити час, витрачений на написання повторюваних елементів коду, автоматизувати типові задачі на зразок валідації форм або формування запитів до бази даних, а також знизити ризик виникнення синтаксичних помилок. Таким чином, використання Copilot суттєво підвищує ефективність праці розробника.

Вбудований у VS Code термінал надає змогу запускати команди Laravel Artisan, Composer, Docker, а також здійснювати міграції, тести та інші операції без необхідності перемикання між програмами. Це сприяє пришвидшенню циклу розробки та покращує загальний досвід роботи.

2. Laravel 12 — це відкритий фреймворк для розробки веб-додатків, який базується на мові програмування PHP та реалізує шаблон проектування Model-View-Controller (MVC). Актуальна версія на момент розробки — Laravel 12 — відзначається стабільністю, покращеною продуктивністю та підтримкою новітніх стандартів мови PHP (зокрема PHP 8.4).

Laravel побудований на основі гнучкої архітектури, яка дозволяє легко масштабувати застосунок та дотримуватись принципів чистого коду (Clean Code) та розділення обов’язків. Він забезпечує чіткий розподіл між логікою додатку (контролери), представленням (Blade-шаблони) та роботою з базами даних (моделі через ORM Eloquent).

Фреймворк також підтримує інверсію залежностей (Dependency Injection), контейнери сервісів, події та обробники, що робить його потужним інструментом для реалізації складних бізнес-процесів.

Laravel має зручну декларативну систему маршрутизації, яка дозволяє визначати маршрути в єдиному файлі (web.php або api.php). Система підтримує RESTful-маршрути, параметри в URL, middleware для авторизації тощо.

Eloquent ORM- це об’єктно-реляційний мапер, що забезпечує зручну роботу з базами даних. Моделі Eloquent дозволяють працювати з таблицями як з об'єктами,

реалізовуючи зв'язки типу "один-до-багатьох", "багато-до-багатьох", поліморфізм та кастомні запити без написання сирого SQL-коду.

Blade — шаблонізатор - це власний двигун шаблонів Blade дозволяє створювати гнучкі HTML-шаблони з логікою (умови, цикли, вставки змінних) у простій синтаксичній формі. Це забезпечує чистоту представлення та полегшує його розширення.

Валідація даних. Laravel пропонує декларативну валідацію форм із можливістю винесення правил у окремі класи FormRequest, що покращує читабельність коду та забезпечує повторне використання логіки перевірки даних.

Для керування структурою бази даних Laravel використовує міграції, які зберігаються у вигляді PHP-класів. Вони дозволяють відслідковувати зміни в БД та зручно розгортати структуру на різних середовищах. Сидери використовуються для автоматичного заповнення таблиць тестовими або базовими даними.

Laravel дозволяє створювати повноцінні RESTful API з використанням ресурсів (API Resource Classes), трансформерів та механізмів авторизації через токени (JWT, Sanctum, Passport).

3. JavaScript — технологія для реалізації клієнтської логіки

У межах розробки вебсайту з ремонту та пошиву одягу JavaScript використовувався як засіб реалізації інтерактивної поведінки елементів інтерфейсу без залучення сторонніх бібліотек або фреймворків (тобто у вигляді чистого або так званого *Vanilla JavaScript*). Основною сферою його застосування стало керування модальними вікнами, що забезпечує більш зручну та динамічну взаємодію користувача з вебінтерфейсом.
Скрипти виконують низку функцій:

Відображення зображень у модальному вікні. При натисканні на відповідний елемент інтерфейсу викликається функція, яка встановлює атрибут src для <img> у модальному блоці та змінює класи Tailwind CSS (hidden/flex) для візуалізації вікна.

Ініціалізація модального вікна скасування дії. Обробник події click відкриває відповідне модальне вікно, а подальше натискання поза межами активного блоку ініціює його закриття. Це забезпечує інтуїтивно зрозумілий механізм взаємодії з користувачем.

Реалізація анімованого відкриття/закриття модального вікна встановлення ціни. Даний компонент супроводжується CSS-анімаціями масштабування (scale-90 → scale-100) та прозорості (opacity-0 → opacity-100), які активуються за допомогою setTimeout, що створює ефект плавної появи інтерфейсу.

Автоматичне фокусування на елементі введення при відкритті модального вікна, що оптимізує користувацький досвід (UX) за рахунок зменшення кількості необхідних дій з боку користувача.

З технічної точки зору, вся функціональність реалізована із використанням подій DOM (DOMContentLoaded, click) та маніпуляцій зі структурою документа через getElementById, classList та інші вбудовані API браузера. Завдяки цьому забезпечується висока продуктивність, низьке навантаження на клієнтську частину та незалежність від зовнішніх залежностей.

Таким чином, JavaScript у цьому проєкті відіграє роль інструменту для управління динамічним інтерфейсом, забезпечуючи зручну та адаптивну взаємодію користувача з вебсайтом.

4. MailTrap — сервіс тестування email-розсилок

MailTrap використовується для безпечної перевірки надсилання електронних листів у середовищі розробки. З його допомогою реалізовано функцію підтвердження електронної пошти користувача при реєстрації. Листи не доходять до реальних користувачів, але повністю імітують роботу поштового сервера, що дозволяє налагодити логіку email-повідомлень.

5. GitHub Copilot — інструмент автоматичної генерації коду

GitHub Copilot — це AI-асистент на основі моделі OpenAI Codex, що інтегрується у редактор коду і пропонує варіанти кодування в реальному часі. У рамках цього проєкту він використовувався для прискоре0ння написання PHP-, JavaScript- та SQL-коду, а

також для генерації повторюваних структур (наприклад, моделей, контролерів, валідаційних класів).

6. DBngin — інструмент для локального запуску СУБД

DBngin — це зручна утиліта для запуску локальних серверів баз даних (MySQL, PostgreSQL тощо) без необхідності складної конфігурації. Вона була використана для швидкого налаштування локального середовища MySQL, що значно пришвидшило процес розробки та тестування запитів.

7. TablePlus — інструмент візуального перегляду та керування базою даних

TablePlus — це клієнт для роботи з базами даних, що дозволяє переглядати таблиці, виконувати SQL-запити, редагувати записи та структуру бази у зручному графічному інтерфейсі. У проєкті він застосовувався для моніторингу стану бази даних під час розробки та налагодження.

8. MySQL — система керування реляційними базами даних

У процесі розробки вебсайту для сервісу ремонту та індивідуального пошиву одягу використовувалася система керування базами даних **MySQL** — одна з найпоширеніших та надійних реляційних СКБД з відкритим вихідним кодом. Вона забезпечує ефективне зберігання, обробку та маніпуляцію структурованими даними у вигляді таблиць, які мають між собою логічні зв’язки.

Для зручного керування базою даних використовувався графічний інтерфейс TablePlus, що забезпечує наочний перегляд таблиць, структури, зв’язків, а також дозволяє виконувати SQL-запити, відлагоджувати та змінювати дані вручну у зручному середовищі.

3.2. Схема реалізації та логіка роботи сайту

Розробка веб-застосунку для онлайн-замовлення ремонту та пошиття одягу здійснювалася за класичною схемою моделі MVC (Model–View–Controller)(рис.3.2), що реалізована у фреймворку Laravel 12. Сайт передбачає обов’язкову реєстрацію та верифікацію користувачів перед оформленням замовлень.

Сайт реалізовано як **веб-застосунок**, який дозволяє зареєстрованим і верифікованим користувачам створювати онлайн-замовлення на пошиття або ремонт одягу. Основною умовою доступу до функціоналу є підтвердження електронної пошти користувачем після реєстрації.

Основні етапи реалізації логіки сайту:

Реєстрація користувача. Користувач створює обліковий запис, вказуючи email та пароль. Реєстрація реалізована за допомогою вбудованих механізмів Laravel Auth.

Підтвердження email (валідація). Після реєстрації система надсилає користувачу email із посиланням для підтвердження. Доступ до створення замовлення надається **лише після успішної верифікації електронної пошти**.

Авторизація.Після підтвердження користувач входить у систему. Його сесія керується стандартними засобами Laravel.

Створення замовлення. Увійшовши до свого облікового запису, користувач має можливість оформити замовлення. Форма замовлення включає:

- назву або тип замовлення (ремонт, пошиття),
- короткий опис,
- суму, яка буде сплачена **готівкою** після виконання послуги.

Збереження замовлення у базу даних. Після заповнення форми дані проходять валідацію (наявність полів, числове значення суми тощо) і зберігаються у базу даних MySQL. Для керування замовленнями використовуються Eloquent-моделі Laravel.

Перегляд замовлень. Користувач може переглядати список своїх замовлень у відповідному розділі особистого кабінету. Це реалізовано через запити до бази даних івивід результатів у Blade-шаблонах.

Рисунок 3.2. Блок Схема MVC (Model–View–Controller).

Результат

Результат

4 ПРАКТИЧНА ЧАСТИНА

4.1. Етапи реалізації функціоналу веб-застосунку

Одним із початкових етапів при створенні веб-додатку є встановлення мови серверного програмування. У даній роботі обрана мова PHP, яка широко використовується для розробки динамічних веб-ресурсів. Її популярність обумовлена низьким порогом входу, широким функціоналом, а також підтримкою великої кількості фреймворків, зокрема Laravel.

1. Завантаження дистрибутива. Версію 8.4.4 PHP було завантажено з офіційного сайту https://www.php.net/downloads.php . Для сумісності з операційною системою Windows обрано архівну версію у форматі .zip, яка містить збірку **Thread Safe**. Такий тип збірки рекомендований для використання з локальними веб-серверами.

Установка та початкове налаштування. Розпакування архіву виконано у каталог C:\php, після чого здійснено налаштування системного середовища. Зокрема, шлях до каталогу було додано до змінної середовища Path, що дозволяє запускати інтерпретатор з будь-якого місця через командний рядок.

Для перевірки коректності встановлення у командному рядку (CMD) виконано таку команду: php -v (Рис. 4.1)

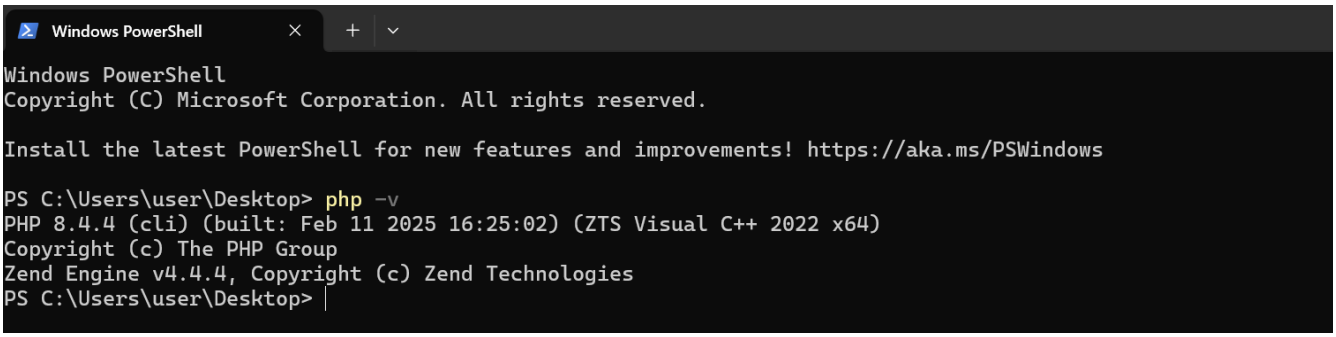

Рисунок 4.1 –Успішне встановлення PHP 8.4.4

2. Встановлення Composer як засобу керування залежностями

Для ефективної роботи з бібліотеками та сторонніми пакетами в екосистемі PHP необхідно використовувати менеджер залежностей. У цій дипломній роботі застосовується **Composer**, який є стандартним інструментом для завантаження та оновлення зовнішніх бібліотек у PHP-проєктах.

Метод встановлення. Інсталяція Composer здійснювалася найпростішим способом — шляхом завантаження офіційного інсталятора з вебсайту https://getcomposer.org. Після запуску інсталяційного файлу всі необхідні налаштування були виконані автоматично. Це дозволило уникнути складної конфігурації та забезпечити готовність інструменту до подальшого використання одразу після завершення встановлення.

Перевірка функціональності. Після інсталяції було проведено базову перевірку роботи Composer. Для цього в командному рядку виконано команду: composer –version (Рис. 4.2).

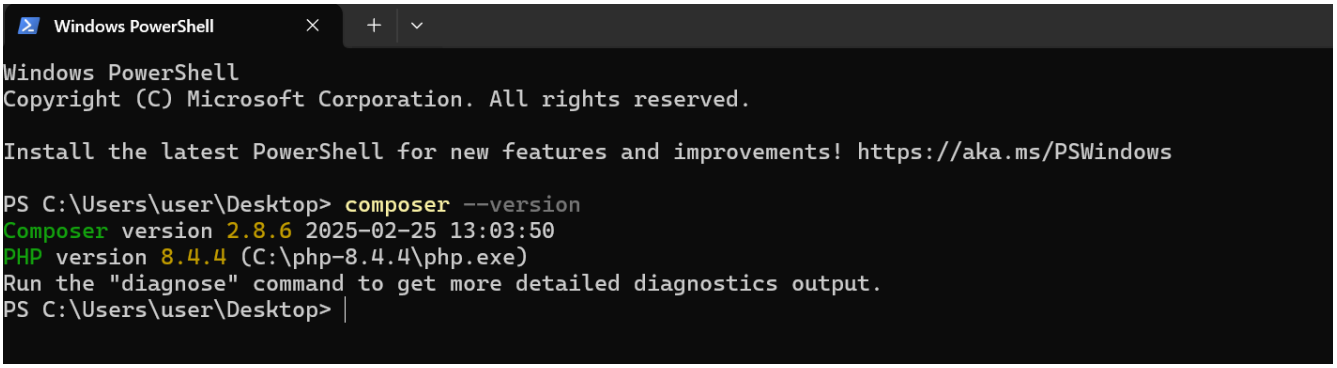

Рисунок 4.2 Версія Composer

У результаті на екрані відобразилася поточна версія встановленого Composer, що підтвердило успішність процедури та готовність середовища до встановлення фреймворку Laravel, а також інших залежностей, необхідних для реалізації функціоналу веб-додатку.

Роль Composer у реалізації проєкту. Composer виконує важливу роль у процесі розробки сайту: з його допомогою будуть встановлені всі необхідні пакети, включно з фреймворком Laravel. Він забезпечує централізоване керування залежностями, автоматичне оновлення бібліотек, а також веде файл конфігурації composer.json, що спрощує спільну розробку та перенесення проєкту на інші середовища.

3. Встановлення фреймворку Laravel 12

Для реалізації серверної частини сайту було обрано сучасний фреймворк Laravel — потужний інструмент для створення веб-застосунків на мові PHP, який поєднує гнучкість, масштабованість та зручність використання. У даному проєкті використано версію Laravel 12, що на момент розробки є актуальною та стабільною.

Ініціалізація нового проєкту

Процес встановлення фреймворку розпочато за допомогою глобального інсталятора Laravel, який був попередньо встановлений у системі. Для створення нового проєкту в командному рядку було введено наступну команду:

laravel new sewing-corner

У результаті виконання команди було автоматично створено нову директорію з назвою sewing-corner, яка містить повну структуру Laravel-застосунку разом із

необхідними залежностями. Завдяки глобальному інсталятору весь процес установки відбувається швидко, без потреби в ручній конфігурації чи додаткових командах.

4. Підключення фронтенд-залежностей та збірка ресурсів

Після встановлення фреймворку Laravel наступним кроком стала інсталяція фронтенд-залежностей та проведення початкової збірки ресурсів інтерфейсу користувача. Laravel 12 використовує сучасний JavaScript-бандлер Vite, який значно пришвидшує обробку CSS, JS, компонентів і забезпечує зручний процес розробки.

Встановлення залежностей

Оскільки Laravel інтегрує інструменти фронтенду, було виконано інсталяцію необхідних пакетів за допомогою менеджера пакетів **npm (Node Package Manager)**. У кореневій директорії проєкту виконано команду: **npm install**

Ця команда завантажує всі залежності, вказані у файлі package.json, включно з Vite, Tailwind CSS, PostCSS, Autoprefixer та іншими бібліотеками, які використовуються для збирання інтерфейсу.

Збірка ресурсів для продакшну

Після встановлення залежностей було здійснено **збірку CSS- та JavaScript-ресурсів** у режимі продакшн. Це дозволяє отримати оптимізовані, мініфіковані файли для подальшого використання у проєкті:

npm run build

Ця команда генерує папку public/build із зібраними ресурсами, які Laravel буде використовувати при рендерінгу сторінок. Продакшн-збірка необхідна для досягнення високої швидкості завантаження сторінок, меншого розміру файлів та покращення продуктивності у реальному середовищі.

На цьому етапі було завершено налаштування клієнтської частини. Усі необхідні залежності підключені, а ресурси зібрані у фінальному вигляді. Це дало можливість перейти до запуску самого сайту у локальному середовищі та подальшої розробки функціоналу.

5. Налаштування бази даних та початкові міграції

Для збереження даних веб-застосунку використовується **реляційна база даних MySQL**, що забезпечує надійність, масштабованість та широкі можливості при роботі з великими обсягами інформації. Laravel містить вбудовану підтримку роботи з MySQL, а налаштування параметрів з'єднання відбувається у конфігураційному файлі .env.

Визначення параметрів підключення

У файлі .env що знаходиться в корені проєкту, було встановлено такі параметри для підключення до бази даних:

DB_CONNECTION=mysql

DB_HOST=127.0.0.1

DB_PORT=3307

DB_DATABASE=sewing-corner

DB_USERNAME=root

DB_PASSWORD=

Тут вказано, що застосунок буде використовувати MySQL як СУБД, підключаючись до локального сервера на порту 3307. Базу даних створено з назвою sewing-corner, користувачем вказано root, без пароля.

Виконання початкових міграцій

Laravel підтримує міграції — механізм для управління структурою бази даних за допомогою коду. Для створення стандартних таблиць, таких як users, password_resets, migrations, було використано команду:

php artisan migrate

У результаті виконання цієї команди фреймворк створює всі необхідні таблиці у зазначеній базі даних відповідно до описаних схем у директорії database/migrations.

Переваги використання міграцій

Використання системи міграцій дозволяє:

- автоматизувати процес створення структури БД;
- контролювати зміни структури під час командної роботи;
- швидко розгортати проєкт на іншому сервері або середовищі.

Таким чином, на цьому етапі структура бази даних була повністю підготовлена до подальшої розробки функціоналу, зокрема створення моделей, таблиць, форм реєстрації та аутентифікації.

6. Запуск проєкту та підготовка фронтенд-ресурсів

Після створення нового Laravel-проєкту наступним кроком стало збирання та запуск клієнтської частини веб-застосунку. Laravel 12 за замовчуванням використовує сучасний інструмент збірки **Vite**, що забезпечує швидку компіляцію CSS, JavaScript та інших фронтенд-ресурсів у процесі розробки.

Запуск фронтенд-збірки.У Laravel стандартні скрипти для роботи з Vite вже заздалегідь визначені у файлі composer.json. Тому для запуску фронтенд-частини (Рис. 4.3) проєкту в командному рядку було виконано:

composer run dev

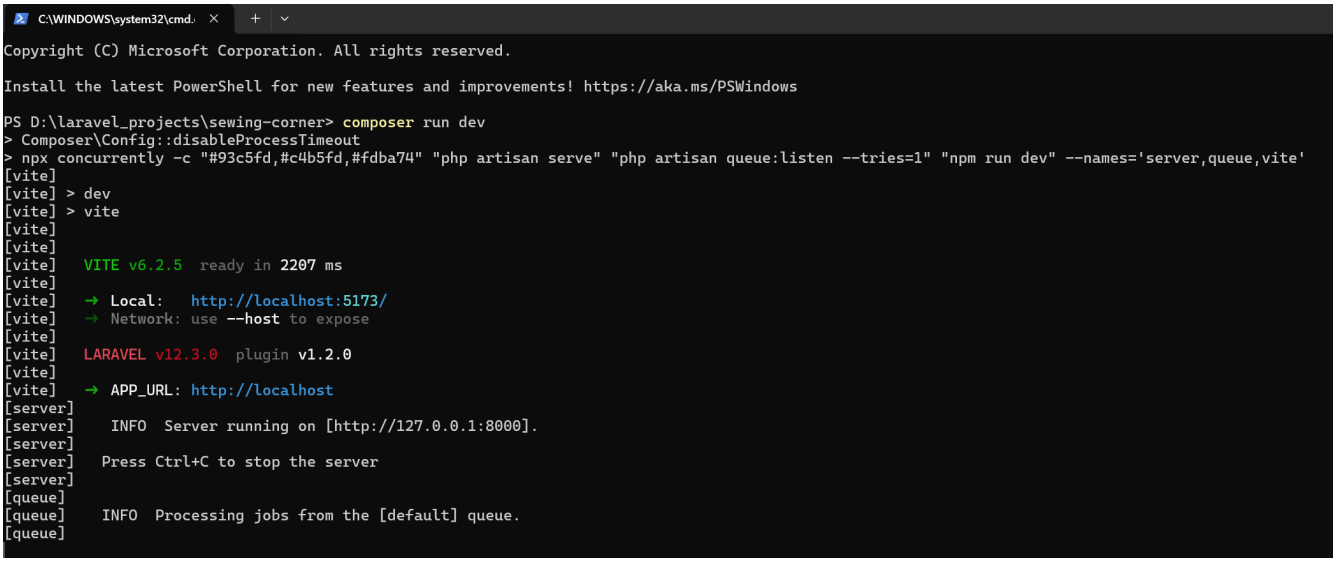

Рисунок 4.3 Запущена збірка

Ця команда активує відповідний сценарій, який викликає Vite для обробки ресурсів. Після запуску Laravel автоматично компілює стилі та скрипти, що дозволяє миттєво застосовувати зміни під час розробки інтерфейсу.

7. Налаштування середовища керування базою даних за допомогою DBngin і TablePlus

Для забезпечення зручного управління базою даних під час розробки програмного забезпечення було використано спеціалізовані інструменти — DBngin та TablePlus. Їх застосування дозволило уникнути складної ручної конфігурації серверів СУБД і значно спростило візуалізацію структури бази даних, перегляд таблиць і взаємодію з ними.

Запуск MySQL-сервера через DBngin

DBngin — це програмне забезпечення, призначене для швидкого створення і запуску локальних інстансів серверів баз даних. У межах реалізації дипломного проєкту було створено інстанс MySQL наступної конфігурації:

- Система управління базами даних: MySQL 8.4.2;
- Порт: 3307;
- Локальний хост: 127.0.0.1.

Обраний порт (3307) дозволив уникнути конфлікту з можливими іншими активними сервісами, які могли використовувати стандартний порт 3306. Сервер MySQL було активовано через інтерфейс DBngin, після чого він став доступним для з’єднання з веб-застосунком Laravel.

Керування базою даних у TablePlus

TablePlus — це клієнтський застосунок для роботи з базами даних, який забезпечує графічний інтерфейс для перегляду, створення та редагування таблиць і записів. Було налаштовано нове з’єднання з базою даних sewing-corner, що розгорнута на локальному сервері MySQL, з такими параметрами:

- Хост: 127.0.0.1;
- Порт: 3307;
- Користувач: root;
- Пароль: відсутній (порожнє поле).

Після встановлення з’єднання було підтверджено наявність таблиць, створених шляхом виконання міграцій, зокрема users, password_reset_tokens, migrations та ін.

Обґрунтування вибору інструментів

Використання DBngin та TablePlus дозволило:

- швидко розгорнути сервер бази даних без командного рядка;
- забезпечити наочний контроль за структурою БД;
- пришвидшити процес розробки та тестування запитів;
- зменшити ризик технічних помилок при роботі з SQL.

8. Інтеграція Tailwind CSS у проєкт

Для створення адаптивного та сучасного дизайну інтерфейсу у проєкті було обрано фреймворк **Tailwind CSS**. Інсталяція виконувалася за офіційною інструкцією на сайті Tailwind:

За допомогою менеджера пакетів npm до проєкту додано Tailwind CSS та офіційний плагін для Vite:

npm install tailwindcss @tailwindcss/vite

Це забезпечило наявність всіх необхідних модулів для обробки стилів у процесі розробки.

У конфігураційному файлі vite.config.ts було підключено плагін @tailwindcss/vite (Рис. 4.4), що дозволило Vite автоматично збирати CSS-утиліти Tailwind під час запуску команди composer run dev.

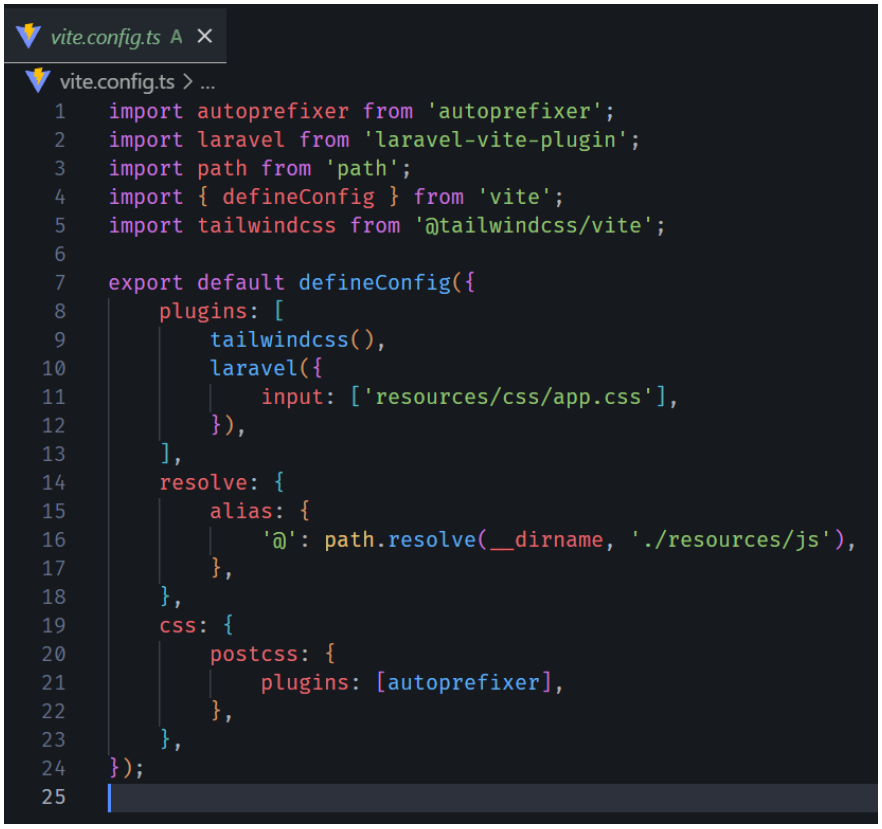

Рисунок 4.4 vite.config.ts

У файлі resources/css/app.css імпортовано tailwind деректівою (Рис. 4.5):

@import "tailwindcss";

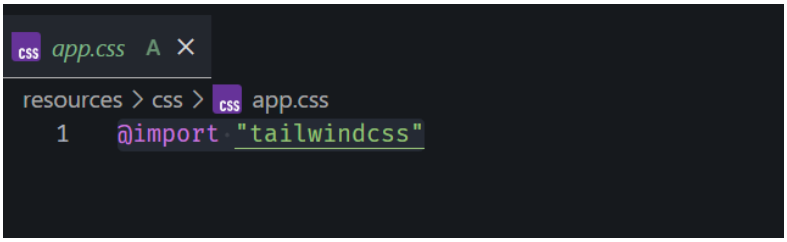

Рисунок 4.5 app.css

Це забезпечує автоматичний підключення всіх базових стилів і утиліт Tailwind CSS згідно з конфігурацією Vite.

4. Конфігурація контент-сканування в tailwind.config.js (Рис. 4.6):

content:

[

'./resources/**/*.blade.php',

'../../vendor/laravel/framework/src/Illuminate/Pagination/resources/views/*.blade.php',

],

```
tailwind.config.js A
tailwind.config.js > ...
1   /** @type {import('tailwindcss').Config} */
2   export default {
3       darkMode: ['class'],
4       content: [
5           './resources/**/*.blade.php',
6           '../../vendor/laravel/framework/src/Illuminate/Pagination/resources/views/*.blade.php',
7
8       ],
9       theme: {···
75      },
76      plugins: [
77          require('@tailwindcss/forms'),
78          require('tailwindcss-animate'),
79      ],
80  };
81
```

Рисунок 4.6 tailwind.config.js

Скомпільований CSS підключено у головному шаблоні через ('resources/css/app.css'), що дозволяє застосовувати утилітарні класи Tailwind безпосередньо в Blade-файлах. Для уніфікації загального вигляду веб-застосунку було створено головний макет **layout.blade.php**, який виконує роль каркасу для всіх сторінок сайту. У цьому файлі зосереджено спільні елементи – підключення стилів через директиву @vite('resources/css/app.css') (Рис. 4.7).

```
layout.blade.php M
resources > views > components > layout.blade.php > html > head
1   <!DOCTYPE html>
2   <html lang="{{ str_replace(search: '_', replace: '-', subject: app()->getLocale()) }}">
3       <head>
4           <meta charset="utf-8">
5           <meta name="viewport" content="width=device-width, initial-scale=1">
6
7           <title>{{ config(key: 'app.name')}}</title>
8
9           <link rel="icon" href="{{ asset(path: 'favicon.ico') }}" type="image/x-icon">
10          <link rel="shortcut icon" href="{{ asset(path: 'favicon.ico') }}" type="image/x-icon">
11
12          @vite(entrypoints: 'resources/css/app.css')
13      </head>
14
15      <body class="flex flex-col h-screen max-w-7xl max-h-[1200px] mx-auto overflow-auto bg-gradient-to-l from-violet-2
16
17          <x-header/>
18
19          <main class="flex-grow">{{ $slot }}</main>
20
21          <x-footer/>
22
23      </body>
24  </html>
```

Рисунок 4.7 layout.blade.php

Директива була додана в <head>. Це забезпечує автоматичне підключення скомпільованого файлу CSS, згенерованого Vite, та дозволяє застосовувати утилітарні класи Tailwind на всіх сторінках, які наслідують цей макет.

Створення маршрутів (Route)

У корені файлу web.php визначено маршрут на головну сторінку:

```
Route::get('/', [IndexController::class, '__invoke'])->name('index');
```

Цей єдиний маршрут делегує обробку invokable-методу контролера IndexController, який вибирає останні замовлення з бази та передає їх у представлення. Подібним чином, за допомогою конструкту Route::view, без створення окремих контролерів, повертаються статичні сторінки:

```
Route::view('/about',   'pages.about')->name('about');

Route::view('/contacts','pages.contact')->name('contact');

Route::view('/terms',   'pages.terms')->name('terms');

Route::view('/privacy', 'pages.privacy')->name('privacy');
```

Таким чином досягається легкість підтримки контенту, який не потребує динамічної логіки.

Адміністративні маршрути згруповано через middleware auth, verified та власний admin, що гарантує виконання тільки автентифікованим користувачам із підтвердженою поштою та роллю адміністратора:

```
Route::middleware(['auth', 'verified', 'admin'])->group(function () {

   Route::get('/admin/dashboard', [AdminController::class, 'dashboard'])-
>name('admin.dashboard');
```

```
    Route::get('/admin/list',          [AdminController::class, 'index'])->name('admin.list');
    Route::post('/admin/orders/{order}/status',
        [AdminController::class, 'updateOrderStatus'])->name('admin.orders.updateStatus');
    Route::put('/orders/{order}',      [OrderController::class, 'update'])->name('orders.update');
    Route::post('/orders/{order}/set-price',
        [OrderController::class, 'setPrice'])->name('orders.setPrice');
    Route::put('/orders/{order}/update-payment-type',
        [OrderController::class, 'updatePaymentType'])->name('orders.updatePaymentType');
});
```

Методи AdminController відповідають за виведення панелі та списку користувачів, а через OrderController адміністратор може змінювати статус, вартість та спосіб оплати замовлення. Така сегментація дозволяє тримати бізнес-логіку безпеки розмежованою від контролерів для звичайних користувачів.

Маршрути, захищені лише auth і verified, обслуговують функціонал зареєстрованого користувача:

```
Route::middleware(['auth', 'verified'])->group(function () {
    Route::get('dashboard', [UserController::class, 'dashboard'])->name('dashboard');
    Route::get('create',    [OrderController::class, 'showCreateForm'])->name('create-order');
    Route::post('create',   [OrderController::class, 'store'])->name('store-order');

    Route::controller(OrderController::class)
        ->prefix('orders')
```

```
        ->name('orders.')
        ->group(function () {
            Route::get('/',        'index')->name('index');
            Route::get('/{order}', 'show')->name('show');
            Route::post('/',       'store')->name('store');
            Route::post('/{order}/cancel','cancel')->name('cancel');
        });
    Route::get('/orders/{order}/edit', [OrderController::class, 'edit'])->name('orders.edit');

    Route::get('/orders/{order}/payment', [PaymentController::class, 'showPaymentForm'])->name('orders.payment');
    Route::get('/profile/edit',   [UserController::class, 'edit'])->name('profile.edit');
    Route::put('/profile/update', [UserController::class, 'update'])->name('profile.update');
});
```

Метод dashboard у UserController повертає персональну панель користувача. Через групу префікса orders інкапсулюється увесь CRUD для замовлень: перелік, перегляд, створення, скасування та редагування. PaymentController забезпечує відображення форми оплати. Роутинг тут чітко відділяє ресус «замовлення» від інших функцій, полегшуючи їхнє розширення й тестування.

Крім того, middleware auth із підписаними URL використовується для верифікації електронної пошти:

```
Route::middleware('auth')->group(function () {
```

```
    Route::get('/email/verify/{id}/{hash}', function (EmailVerificationRequest $request) {
        $request->fulfill();
        return redirect()->route('dashboard');
    })->middleware('signed')->name('verification.verify');

    Route::post('/email/verification-notification', function (Request $request) {
        $request->user()->sendEmailVerificationNotification();
        return back()->with('message', 'Verification link sent!');
    })->middleware(['throttle:3,1'])->name('verification.send');

    Route::get('/verify-email', function () {
        return view('user.verify-email');
    })->name('verification.notice');

    Route::post('logout', [UserController::class, 'logout'])->name('logout');
});
```

Тут підписані посилання гарантують автентичність запиту на верифікацію, а лімітер throttle запобігає надмірним повторним відправкам електронного листа.

Гостьові маршрути обмежені middleware guest, що унеможливлює їхнє використання авторизованими користувачами:

```
Route::middleware('guest')->group(function () {
```

```
Route::get('register',    [UserController::class, 'showRegistrationForm'])->name('register');
Route::post('register',   [UserController::class, 'store'])->name('register');
Route::get('login',       [UserController::class, 'showLoginForm'])->name('login');
Route::post('login',      [UserController::class, 'login'])->name('login');
Route::get('forgot-password', function () { return view('user.forgot-password'); })->name('password.request');
Route::post('forgot-password', [UserController::class, 'forgotPasswordStore'])
    ->name('password.email')->middleware('throttle:3,1');
Route::get('reset-password/{token}', function (string $token) {
    return view('user.reset-password', ['token' => $token]);
})->name('password.reset');
Route::post('reset-password', [UserController::class , 'resetPasswordUpdate'])->name('password.update');
});
```

Ця секція обробляє реєстрацію, вхід, ініціацію та скидання паролю. Використання throttle на маршруті відправки листа забезпечує захист від бот-атак.

Створення Кнотролерів:

IndexController, розташований у просторі імен App\Http\Controllers, реалізує єдиний метод-інвокацію __invoke, що робить його викликаним як функціональний об’єкт. У цьому методі здійснюється запит до моделі Order з використанням ланцюгової побудови Eloquent-запиту: Order::latest()->take(4)->get(). Спочатку через latest() отримуються записи в порядку спадання дати створення, далі обмеження take(4) гарантує вибірку чотирьох найсвіжіших замовлень, після чого виконується фактичне

завантаження даних методом get(). Результат передається у подання pages.index за допомогою функції view(), а змінна $orders інкапсулюється в масиві параметрів через compact('orders').

Таким чином, **IndexController** відповідає за підготовку та передачу у вигляд інформації про останні чотири замовлення, делегуючи власне відображення шаблону—це чітко розділяє роль контролера (отримання даних і побудова моделі представлення) і представлення (рендеринг HTML).

OrderController виступає єдиним точковим входом для керування замовленнями в системі та інкапсулює всю бізнес-логіку, пов’язану з їхнім відображенням, створенням, редагуванням і оплатою.

Метод **index** приймає запит з можливим параметром search і, залежно від ролі поточного користувача, повертає пагіновану колекцію замовлень. Якщо роль адміністратора, виконується

```
$orders = Order::latest()
    ->when($request->search, function ($query, $search) {
        $query->where('service_type', 'like', "%{$search}%")
            ->orWhere('description', 'like', "%{$search}%");
    })
    ->paginate(6)
    ->withQueryString();
```

— таким чином адміністратор бачить усі заявки з можливістю фільтрації за типом послуги чи описом. У протилежному випадку формується запит

```
$orders = Order::where('user_id', auth()->id())
```

```
    ->latest()
    ->when($request->search, /* ... */)
    ->paginate(6)
    ->withQueryString();
```

що гарантує, що звичайний користувач бачить лише власні замовлення.

Метод **show** приймає модель Order через route-model binding і перевіряє, чи має поточний користувач доступ до перегляду цього запису. Якщо роль не адміністратора й ідентифікатор власника не співпадає з ідентифікатором залогіненого, викликається

```
abort(403, 'У вас немає доступу до цього замовлення.');
```

інакше виконується

```
return view('pages.orders.show', compact('order'));
```

Метод **showCreateForm** повертає представлення форми створення, а **store** спочатку валідовує вхідні дані

```
$validated = $request->validate([
    'service_type' => 'required|string',
    'description'  => 'required|string',
    'images'       => 'required|array|min:2',
    'images.*'     => 'image|mimes:jpeg,png,jpg,gif|max:20480',
]);
```

потім створює новий об’єкт Order зі статусом new і зберігає його в базі. Після цього обробляються файли зображень, кожне з яких завантажується в директорію image-order у сховищі public, шляхи зберігаються в полі images у форматі JSON.

Метод cancel приймає об’єкт Request та ідентифікатор замовлення, після чого виконує пошук в базі через Order::findOrFail($orderId). Якщо статус замовлення вже дорівнює рядку "completed", виклик redirect()->back()->with('error', …) повертає користувачеві повідомлення про неможливість скасування. Інакше контролер перевіряє, чи поточний користувач є власником замовлення ($order->user_id === auth()->id()) або має роль адміністратора (auth()->user()->role === UserRole::Admin), в іншому випадку викликається abort(403). Після проходження валідації полі статусу встановлюється в "canceled", виконується $order->save(), і клієнт перенаправляється назад із флеш-повідомленням успіху:

```
public function cancel(Request $request, $orderId)
{
    $order = Order::findOrFail($orderId);

    if ($order->status === 'completed') {
        return redirect()->back()->with('error', 'Неможливо скасувати виконане замовлення!');
    }

    if ($order->user_id !== auth()->id() && auth()->user()->role !== UserRole::Admin) {
        abort(403, 'У вас немає доступу до скасування цього замовлення.');
    }

    $order->status = 'canceled';
    $order->save();

    return redirect()->back()->with('success', 'Замовлення успішно скасовано!');
}
```

Метод edit приймає модель Order і за допомогою тих самих перевірок прав доступу (abort(403) для неавторизованих) повертає представлення pages.orders.edit разом із даними замовлення. У методі update повторюються аутентифікаційні перевірки, далі відбувається валідація вхідної інформації:

```
$validated = $request->validate([
    'service_type' => 'required|string',
    'description'  => 'required|string|max:150',
    'images.*'     => 'nullable|image|mimes:jpeg,png,jpg,gif|max:20480',
    'status'       => 'nullable|in:new,processing,completed,canceled',
]);
```

Після цього об’єкт Order оновлюється полями service_type і description. Якщо користувач має роль адміністратора і в запиті передано новий status, він також застосовується. Нові файли зображень, якщо є, завантажуються в публічний диск через метод store, шляхи кодуються в JSON та зберігаються в полі images. Завершується операція викликом $order->save() та перенаправленням на маршрут перегляду замовлення з підтвердженням:

```
public function update(Request $request, Order $order)
{
    if ($order->user_id !== auth()->id() && auth()->user()->role !== UserRole::Admin) {
        abort(403, 'У вас немає доступу до оновлення цього замовлення.');
    }

    $validated = $request->validate([
        'service_type' => 'required|string',
        'description'  => 'required|string|max:150',
```

```
        'images.*'     => 'nullable|image|mimes:jpeg,png,jpg,gif|max:20480',
        'status'       => 'nullable|in:new,processing,completed,canceled',
    ]);

    $order->service_type = $validated['service_type'];
    $order->description  = $validated['description'];

    if (auth()->user()->role === UserRole::Admin && isset($validated['status'])) {
        $order->status = $validated['status'];
    }

    if ($request->hasFile('images')) {
        $paths = array_map(
            fn($img) => $img->store('image-order', 'public'),
            $request->file('images')
        );
        $order->images = json_encode($paths);
    }

    $order->save();

    return redirect()->route('orders.show', $order->id)
                     ->with('success', 'Замовлення успішно оновлено!');
}
```

Метод setPrice обробляє POST-запит на встановлення ціни замовлення. Спершу перевіряється право власності або адміністратора, потім відбувається валідація, яка гарантує, що поле price є невід’ємним числом. Після цього полі order->price

присвоюється нове значення і виконується $order->save(), а клієнт повертається до сторінки перегляду з відповідним повідомленням:

```
public function setPrice(Request $request, Order $order)
{
    if ($order->user_id !== auth()->id() && auth()->user()->role !== UserRole::Admin) {
        abort(403, 'У вас немає доступу до оновлення ціни цього замовлення.');
    }

    $validated = $request->validate([
        'price' => 'required|numeric|min:0',
    ]);

    $order->price = $validated['price'];
    $order->save();

    return redirect()->route('orders.show', $order->id)
                ->with('success', 'Ціна замовлення успішно оновлена!');
}
```

Метод updatePaymentType відповідає за зміну статусу оплати. Якщо замовлення вже позначено як paid, виконання блокується через back()->with('error'). Інакше здійснюється валідація одного поля payment_status, значення якого має належати до дозволеного переліку, після чого оновлене поле зберігається і користувач отримує флеш-повідомлення:

```
public function updatePaymentType(Request $request, Order $order)
{
    if ($order->payment_status === PaymentStatus::Paid) {
```

```
        return back()->with('error', 'Неможливо змінити тип оплати для вже оплаченого замовлення.');
    }

    $validated = $request->validate([
        'payment_status' => 'required|in:pending,cash,failed,paid',
    ]);

    $order->payment_status = $validated['payment_status'];
    $order->save();

    return back()->with('success', 'Тип оплати успішно оновлено.');
}
```

Таким чином, **OrderController** забезпечує повний набір операцій із замовленнями, гарантуючи коректність даних і безпеку доступу, та дотримується архітектурних принципів MVC, де контролер відповідає за обробку запитів, валідацію та перенаправлення.

UserController. У просторі імен App\Http\Controllers відповідає за увесь процес аутентифікації та управління профілем користувача.

Метод showRegistrationForm() просто повертає представлення реєстрації через виклик view('user.register'), після чого користувач бачить форму для введення імені, email, пароля та телефону.

При обробці POST-запиту реєстрації метод store(Request $request) використовує фасад Validator для перевірки коректності даних за правилами, включно з унікальністю email у таблиці users та регулярним виразом для українського номера телефону. У разі валідаційних помилок користувач повертається на форму із збереженими даними та повідомленнями про помилки. Якщо валідація проходить, новий запис створюється

через User::create([...]), пароль хешується методом Hash::make(), а роль за замовчуванням встановлюється як 'user'. Після успішного збереження відбувається автоматичний вхід через Auth::login($user), а користувач перенаправляється на маршрут верифікації електронної пошти з флеш-повідомленням про успіх:

```
$user = User::create([
    'name'     => $request->name,
    'email'    => $request->email,
    'password' => Hash::make($request->password),
    'phone'    => $request->phone,
    'role'     => 'user',
]);
Auth::login($user);
return redirect()->route('verification.notice')
    ->with('success', 'Реєстрація успішна! Ви увійшли в свій кабінет.');
```

Метод dashboard() перевіряє роль поточного користувача через auth()->user()->role. Якщо це UserRole::Admin, відбувається редирект на адміністраторську панель route('admin.dashboard') з флеш-повідомленням, інакше повертається представлення користувацького дашборду view('user.dashboard').

Форма входу відображається через showLoginForm(), яка повертає view('user.login'). Обробка надісланих облікових даних відбувається в методі login(Request $request), де спочатку валідується поле login (ім’я або email) і password. За допомогою функції filter_var(..., FILTER_VALIDATE_EMAIL) динамічно визначається, чи використовувати поле email або name для спроби аутентифікації через auth()->attempt([...], $remember). У разі невдачі користувач повертається назад із загальною помилкою:

```
$loginField = filter_var($request->login, FILTER_VALIDATE_EMAIL) ? 'email' : 'name';

if (auth()->attempt([$loginField => $request->login, 'password' => $request->password], $remember)) {

    return redirect()->route('dashboard')->with('success', 'Вхід успішний!');

}

return redirect()->back()->withErrors([

    'login' => 'Неправильне ім\'я, email або пароль.'

]);
```

Метод forgotPasswordStore(Request $request) ініціює відправку посилання для скидання паролю: після валідації поля email фасад Password::sendResetLink() надсилає листа, і в залежності від статусу (Password::RESET_LINK_SENT або Password::INVALID_USER) повертає флеш-повідомлення або помилку.

У методі resetPasswordUpdate(Request $request) здійснюється безпосереднє оновлення пароля. Після валідації поля token та пароля фасад Password::reset() виконує колбек, у якому через forceFill() встановлюється новий пароль без хешування (якщо потрібно, хешування налаштоване в моделі), генерується новий remember-токен за допомогою Str::random(60), а подія PasswordReset сповіщає систему. Успіх призводить до редиректу на форму входу з повідомленням:

```
$status = Password::reset(

    $request->only('password', 'password_confirmation', 'token'),

    function (User $user, string $password) {

        $user->forceFill([

            'password' => $password
```

```
        ])->setRememberToken(Str::random(60));
        $user->save();
        event(new PasswordReset($user));
    }
);
```

Методи edit() та update(Request $request) працюють із профілем користувача: перший повертає представлення user.edit-profile із поточним об’єктом користувача, другий проводить валідацію полів name, email (з урахуванням виключення поточного запису із унікальності) та phone, а потім оновлює атрибути через $user->update($validated). Для збереження сесії після зміни email виконується auth()->login($user), і користувач повертається на дашборд з повідомленням про успіх.

Нарешті, метод logout(Request $request) викликає auth()->logout(), знищує сесію і перенаправляє користувача на сторінку входу з підтвердженням виходу:

```
public function logout(Request $request)
{
    auth()->logout();
    return redirect()->route('login')->with('success', 'Вихід успішний!');
}
```

Таким чином, **UserController** централізує весь цикл життя аутентифікації та профілю користувача: від реєстрації й входу до керування обліковим записом і скидання пароля, гарантуючи при цьому безпеку за допомогою валідації, захисту від CSRF і подій аутентифікації.

Контролер AdminController інкапсулює базовий адміністративний функціонал: перегляд списку користувачів, відображення власної адмін-панелі та оновлення статусу замовлення.

```
public function index()
{
    $users = User::all(); // Отримуємо всіх користувачів
    return view('user.admin.list', compact('users'));
}
```

Метод index() звертається до моделі User через Eloquent, завантажує колекцію всіх записів із таблиці users і передає її у представлення user.admin.list. У цьому шаблоні формується табличний інтерфейс, у якому адмін може переглядати деталі кожного акаунта.

```
public function dashboard()
{
    return view('user.admin.dashboard');
}
```

Метод dashboard() повертає статичне представлення адмін-панелі user.admin.dashboard, у якому зазвичай відображаються загальні статистичні дані або швидкі посилання на основні адміністративні дії.

```
public function updateOrderStatus(Request $request, $orderId)
{
    $request->validate([
        'status' => 'required|in:new,processing,completed,canceled',
    ]);
    $order = Order::findOrFail($orderId);
    $order->status = $request->status;
    $order->save();
```

```
    return back()->with('success', 'Статус замовлення оновлено!');
}
```

Метод updateOrderStatus() приймає HTTP-запит із полем status і ідентифікатором замовлення. На першому кроці виконується валідація, що гарантує, що новий статус належить до дозволеного переліку. Далі через Order::findOrFail($orderId) завантажується потрібний запис або створюється виключення 404, якщо його немає. Після зміни властивості status об’єкт зберігається (save()), а адмін повертається на попередню сторінку з флеш-повідомленням про успіх.

Таким чином, AdminController надає мінімальний набір дій для адміністратора: перегляд усіх користувачів, доступ до адмінської панелі та можливість змінювати статуси замовлень без дублювання бізнес-логіки в інших контролерах.

Створення Моделей (Model)

Модель **Order** наслідує базовий клас Eloquent Model та використовує трейти HasFactory для підтримки фабрик у тестах і заповнення бази. Через властивість $fillable вказано перелік масово доступних полів — user_id, service_type, description, images, price, status, payment_status і payment_date, що забезпечує безпечне присвоєння цих атрибутів під час викликів create() та update().

Визначений зв’язок із користувачем реалізується методом:

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

Це дозволяє через $order->user завантажувати відповідну модель User згідно з зовнішнім ключем user_id.

У $casts встановлено автоматичне приведення масиву шляхи зображень:

```
'images' => 'array',
```

та перетворення рядкового статусу оплати на відповідний перелік-енумерацію:

'payment_status' => PaymentStatus::class,

а також автоматичний конвертер дати оплати у екземпляр DateTime:

'payment_date' => 'datetime',

—це спрощує роботу з цими атрибутами в коді, дозволяючи отримувати для images вже PHP-масив, для payment_status — об’єкт-енум, і для payment_date — інтерфейс Carbon.

Метод-утиліта **isPaid()** повертає булеве значення, яке свідчить про факт оплати замовлення:

```
public function isPaid(): bool
{
    return $this->payment_status === PaymentStatus::Paid ||
        $this->payment_status === PaymentStatus::Cash;
}
```

Це інкапсулює бізнес-логіку визначення оплаченості, полегшуючи виклики виду if ($order->isPaid()) в контролерах та представленнях. Так, модель **Order** виступає одночасно сховищем даних і носієм пов’язаної предметної логіки, що відповідає принципам активного запису (Active Record) у Laravel.

Модель User Наслідує від базового класу Authenticatable і реалізує контракти MustVerifyEmail та CanResetPassword, що автоматично підключає механізми верифікації електронної пошти та відновлення пароля. За допомогою трейта HasFactory забезпечується підтримка фабрик для тестів, а трейт Notifiable дозволяє надсилати повідомлення (наприклад, email-сповіщення).

class User extends Authenticatable implements MustVerifyEmail, CanResetPassword

Властивість $fillable визначає перелік атрибутів, які можуть заповнюватися масово, зокрема ім’я, email, пароль, телефон і роль користувача:

```
protected $fillable = [
    'name',
    'email',
    'password',
    'phone',
    'role',
];
```

Атрибути password та remember_token заховані від серіалізації, що захищає чутливі дані при перетворенні моделі в масив або JSON:

```
protected $hidden = [
    'password',
    'remember_token',
];
```

Метод casts() забезпечує автоматичне приведення полів: email_verified_at стає об’єктом datetime, password — автоматично хешується при присвоєнні, phone приводиться до рядка, а role інкапсулюється в екземпляр перерахування UserRole:

```
protected function casts(): array
{
    return [
        'email_verified_at' => 'datetime',
```

```
        'password' => 'hashed',
        'phone' => 'string',
        'role' => UserRole::class,
    ];
}
```

Таким чином, модель User об’єднує в собі можливості аутентифікації, підтвердження email, скидання пароля та роботу з фабриками й нотифікаціями, дотримуючись принципів безпеки та типізації в Laravel.

Middleware

AdminMiddleware перехоплює вхідний HTTP-запит і перевіряє роль автентифікованого користувача, гарантуючи, що доступ до адміністративних маршрутів матимуть лише користувачі з роллю Admin. Після отримання запиту фреймворк викликає метод handle(), який виконує таку логіку:

```
public function handle(Request $request, Closure $next): Response
{
    if (auth()->user()->role !== UserRole::Admin) {
        // Якщо роль відмінна від Admin — перенаправляємо на дашборд
        return redirect()->route('dashboard');
    }
    // В іншому випадку передаємо запит далі в ланцюг обробників
    return $next($request);
}
```

Спершу через фасад auth() отримується поточний користувач, а його властивість role, типізована як екземпляр UserRole, порівнюється з константою UserRole::Admin. Якщо перевірка не пройдена, виконується редирект на звичайну користувацьку панель (dashboard). Інакше виклик $next($request) передає контроль наступному шарові — наступному middleware або безпосередньо контролеру. Таке рішення дозволяє централізовано захищати групи маршрутів від доступу неадміністраторів, дотримуючись принципу єдиного місця перевірки прав.

RoleMiddleware забезпечує універсальний механізм контролю доступу за ролями користувача. Його метод handle приймає поточний HTTP-запит, функціональний об’єкт наступного обробника та назву ролі, яку необхідно перевірити:

```
public function handle(Request $request, Closure $next, $role)
{
    if (auth()->check() && auth()->user()->role === $role) {
        return $next($request);
    }
    abort(403, 'У вас немає доступу до цієї сторінки.');
}
```

Спочатку виконується перевірка автентифікації через auth()->check(), а потім порівнюється властивість role моделі користувача з переданим параметром $role. Якщо обидві умови задовольняються, виклик $next($request) передає керування наступному middleware або кінцевому контролеру. Інакше автоматично викликається HTTP-помилка 403 з повідомленням про відсутність доступу. Завдяки динамічному аргументу $role цей middleware можна повторно використовувати для захисту будь-яких маршрутів за різними ролями без дублювання коду.

Enum

Перерахування OrderStatus у просторі імен App\Enums використовує можливості PHP для введення строго типізованих сталих значень, що описують життєвий цикл замовлення. Визначено чотири стани:

```
enum OrderStatus: string
{
    case New        = 'new';
    case Processing = 'processing';
    case Completed  = 'completed';
    case Canceled   = 'canceled';
}
```

Кожен випадок (case) має рядкове представлення, яке зручно зберігати в базі даних і безпосередньо порівнювати в логіці застосунку. Використання PHP-enum з підвищує безпеку та зручність рефакторингу та гарантує, що в коді будуть використані лише визначені стани замовлення, а спроба присвоїти або порівняти зі стороннім значенням призведе до помилки на етапі виконання або валідації коду. Це сприяє підтримці однорідності коду й зменшує ризик логічних помилок при перевірці статусів.

Перерахування PaymentStatus визначає можливі стани оплати замовлення й розташоване в просторі імен App\Enums. Воно наслідує від базового типу string, завдяки чому кожен випадок безпосередньо відображається в базі як рядок:

```
enum PaymentStatus: string
{
    case Pending = 'pending';
    case Paid    = 'paid';
```

```
    case Cash    = 'cash';
    case Failed  = 'failed';
}
```

Значення Pending позначає замовлення, для якого очікується оплата, Paid — повністю сплачене online, Cash — оплата готівкою при отриманні, а Failed — неуспішну оплату.

Перерахування UserRole у просторі імен App\Enums визначає ролі, які можуть бути присвоєні користувачам у системі. Кожен випадок типізовано як рядкове значення, що зберігається в базі даних та використовується в бізнес-логіці:

```
enum UserRole: string
{
    case User  = 'user';
    case Admin = 'admin';
}
```

через динамічні middleware (RoleMiddleware) гарантувати, що тільки користувачі з відповідною роллю отримають доступ до певних частин інтерфейсу або операцій.

Створення Бази даних

Після ініціалізації Laravel-проєкту важливою складовою є налаштування з’єднання з системою керування базами даних. У цьому випадку обрано СКБД MySQL версії 8.4.2 (x64), яка характеризується високою продуктивністю, стабільністю та широкою підтримкою транзакцій. Для коректної інтеграції бази даних із Laravel у конфігураційному файлі .env було визначено ключові параметри: драйвером обрано MySQL, вказано локальну IP-адресу хоста 127.0.0.1, порт з’єднання 3307 (з метою уникнення конфліктів із іншими службами), ім’я бази даних sewing-corner, а також облікові дані користувача — логін root без пароля.

Запуск локального серверу MySQL здійснювався за допомогою утиліти DBngin, яка забезпечує зручне управління окремими інстанціями СКБД. Створено сервер з ідентифікатором sewing-corner, що працює на вказаному порті 3307(Рис. 13). Після цього для візуального адміністрування та взаємодії з БД було використано клієнт TablePlus (Рис. 14). Завдяки йому виконано підключення до раніше створеного серверу, що дозволило перевірити доступність бази даних і підготувати середовище до подальшої роботи з ORM Laravel.

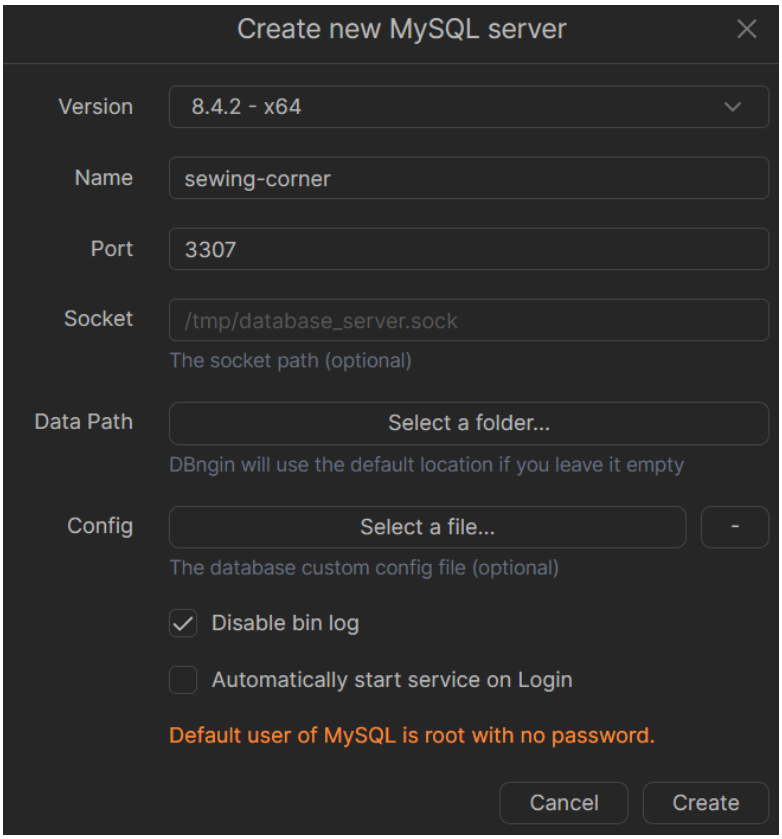

Рисунок 13 – Створення MySQL сервера в DBngin

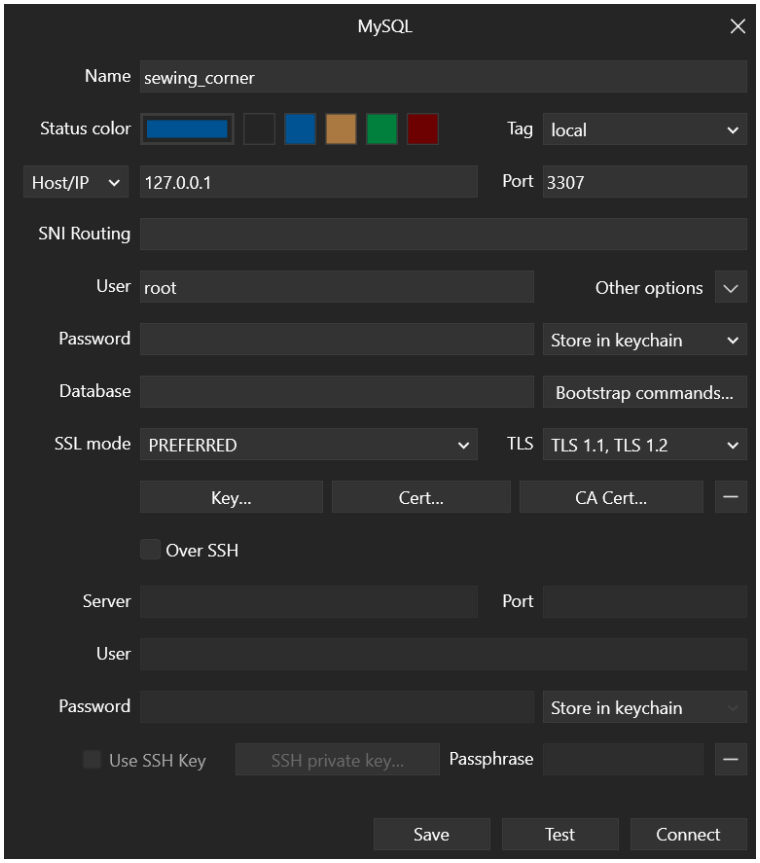

Рисунок 14 – З’єднання Table Plus з базою sewing-corner

Наступним етапом після встановлення з’єднання є формування структури бази даних. Laravel надає механізм міграцій, що дозволяє задавати схеми таблиць безпосередньо у коді, використовуючи PHP-інтерфейс. Це сприяє прозорості змін, які фіксуються у системі контролю версій, і забезпечує відтворюваність архітектури БД у різних середовищах. Команди створення та застосування міграцій дозволяють централізовано керувати еволюцією структури БД, що відповідає сучасному підходу до інфраструктури як коду та сприяє стабільному й контрольованому розвитку програмного забезпечення.

Міграція

У початковому етапі розробки Laravel-застосунку важливу роль відіграє базова міграція, яка створюється автоматично після ініціалізації проєкту. Ця початкова міграція визначає мінімально необхідну структуру бази даних для забезпечення функціонування

механізмів аутентифікації, відновлення паролю та управління сесіями користувачів. Вона реалізується за допомогою класу, який розширює базовий абстрактний клас Migration, і складається з двох методів: up() — для створення таблиць, і down() — для їх видалення.

У межах методу up() описано створення трьох основних таблиць. Перша з них — users, що слугує основною для зберігання облікових записів користувачів. Вона містить поля для ідентифікатора (id), імені, унікальної адреси електронної пошти, хешованого пароля, мітки підтвердження електронної пошти, токена запам’ятовування (remember_token) і часових міток створення та оновлення записів. Ця структура дозволяє реалізувати базову реєстрацію, вхід та підтвердження пошти відповідно до механізмів Laravel.

Друга таблиця — password_reset_tokens, що відповідає за зберігання токенів для скидання пароля. Вона включає поля для адреси електронної пошти (яка також є первинним ключем), токена та часу його створення. Завдяки цьому забезпечується безпечний і контрольований процес відновлення доступу до акаунту.

Третя таблиця — sessions, яка використовується для управління сесіями користувачів. Вона зберігає унікальний ідентифікатор сесії, ідентифікатор користувача (з можливістю встановлення зовнішнього ключа), IP-адресу, інформацію про клієнтський браузер (user_agent), сесійну інформацію у форматі payload, а також часову позначку останньої активності. Ця таблиця забезпечує можливість використання драйвера database для зберігання сесій у Laravel, що є особливо корисним у багатосерверних середовищах або при підвищених вимогах до безпеки.

Метод down() забезпечує обернений процес — видалення відповідних таблиць у разі необхідності відкату міграцій. Така модульна та декларативна структура дозволяє централізовано та передбачувано керувати еволюцією бази даних у рамках архітектури Laravel.

```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
return new class extends Migration
{
    public function up(): void
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });

        Schema::create('password_reset_tokens', function (Blueprint $table) {
            $table->string('email')->primary();
            $table->string('token');
```

```
            $table->timestamp('created_at')->nullable();
        });

        Schema::create('sessions', function (Blueprint $table) {
            $table->string('id')->primary();
            $table->foreignId('user_id')->nullable()->index();
            $table->string('ip_address', 45)->nullable();
            $table->text('user_agent')->nullable();
            $table->longText('payload');
            $table->integer('last_activity')->index();
        });
    }
    public function down(): void
    {
        Schema::dropIfExists('users');
        Schema::dropIfExists('password_reset_tokens');
        Schema::dropIfExists('sessions');
    }
};
```

Міграція таблиці замовлень (orders)

У рамках розробки програмного забезпечення було реалізовано міграцію, яка створює таблицю orders, що виступає центральною складовою моделі обліку клієнтських замовлень. Міграція створена із застосуванням засобів фреймворку Laravel, що забезпечує декларативне визначення структури таблиць за допомогою об'єктно-орієнтованого інтерфейсу.

Таблиця orders містить унікальний ідентифікатор запису (id), який автоматично інкрементується, що дозволяє однозначно ідентифікувати кожне замовлення. Поле user_id визначено як ціле беззнакове 64-бітне число та слугує зовнішнім ключем до таблиці users, реалізуючи міжтабличну залежність «багато до одного» (many-to-one) між замовленням і користувачем. Для забезпечення цілісності даних використано обмеження зовнішнього ключа із директивою onDelete('cascade'), що передбачає автоматичне видалення пов’язаних замовлень у випадку видалення відповідного запису користувача.

Поле service_type визначене як рядковий тип і зберігає категорію або тип замовленої послуги. Опис послуги задається у вигляді текстового поля description, яке є опціональним (nullable), що дозволяє гнучко формулювати інформацію щодо конкретного запиту. Поле images представлене у форматі JSON і призначене для зберігання масиву зображень, що супроводжують замовлення, наприклад, фотографій виробу або ескізів. Така структура є оптимальною для сучасних веб-застосунків, де дані часто мають вкладену або колекційну природу.

Поле price зберігає числове значення вартості послуги, визначене як ціле число зі значенням за замовчуванням 0. Поля created_at і updated_at автоматично генеруються засобами Laravel через директиву $table->timestamps(), що дозволяє відслідковувати дату й час створення та оновлення запису.

Таким чином, дана міграція забезпечує створення гнучкої, масштабованої та реляційно зв’язаної структури збереження замовлень, що відповідає вимогам сучасних веб-орієнтованих інформаційних систем:

```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
return new class extends Migration
{
    public function up(): void
    {
        Schema::create('orders', function (Blueprint $table) {
            $table->id();
            $table->unsignedBigInteger('user_id'); // Прив'язка до користувача
            $table->string('service_type'); // Тип послуги
            $table->text('description')->nullable(); // Опис роботи
            $table->json('images')->nullable(); // Зображення у форматі JSON
            $table->integer('price')->default(0); // Ціна (опціонально)
            $table->timestamps();
            // Зовнішній ключ
            $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');
        });
    }
    public function down(): void
    {
        Schema::dropIfExists('orders');
    }
```

```
};
```

У міграції 2025_05_05_080900_add_role_to_users_table.php реалізовано розширення структури таблиці users шляхом додавання нового атрибуту phone, який зберігає номер телефону користувача у вигляді рядка довжиною до 13 символів. Поле є необов’язковим (nullable) і додається після колонки email. У разі відкоту міграції поле phone буде видалене, що забезпечує зворотну сумісність структури бази даних.

```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
return new class extends Migration
{
   public function up(): void
   {
      Schema::table('users', function (Blueprint $table) {
         $table->enum('role',['user', 'admin'])->default('user')->after('phone');
      });
   }
   public function down(): void
   {
      Schema::table('users', function (Blueprint $table) {
         $table->dropColumn('role');
      });
   }
};
<?php
use Illuminate\Database\Migrations\Migration;
```

```
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
return new class extends Migration
{
    public function up(): void
    {
        Schema::table('orders', function (Blueprint $table) {
            $table->string('status')->default('new')->after('id');
        });
    }
    public function down(): void
    {
        Schema::table('orders', function (Blueprint $table) {
            $table->dropColumn('status');
        });
    }
};
```

У міграції 2025_05_12_105151_add_status_to_orders_table.php здійснено модифікацію таблиці orders шляхом додавання нового поля status типу string, яке характеризує поточний стан замовлення. Полю надано значення за замовчуванням — 'new'. Зміна реалізована із дотриманням принципів зворотності: при відкочуванні міграції атрибут буде видалено з таблиці.

```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
```

```
return new class extends Migration
{
    public function up(): void
    {
        Schema::table('orders', function (Blueprint $table) {
            $table->string('status')->default('new')->after('id');
        });
    }
    public function down(): void
    {
        Schema::table('orders', function (Blueprint $table) {
            $table->dropColumn('status');
        });
    }
};
```

Міграція 2025_05_15_213757_update_payment_status_in_orders_table.php розширює структуру таблиці orders у базі даних шляхом додавання нового стовпця payment_status типу enum. Даний стовпець може приймати одне з чотирьох значень: 'pending', 'paid', 'failed', 'cash', що дозволяє зберігати поточний статус оплати для кожного замовлення. Значення за замовчуванням встановлено як 'pending', що забезпечує коректну ініціалізацію статусу для нових записів. У разі відкату міграції стовпець payment_status буде видалено з таблиці orders, повертаючи структуру до попереднього стану. Така зміна

сприяє підвищенню гнучкості та деталізації обліку фінансових операцій у системі керування замовленнями.

```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
return new class extends Migration
{
    public function up(): void
    {
        Schema::table('orders', function (Blueprint $table) {
            $table-> enum('payment_status', ['pending', 'paid', 'failed', 'cash'])->default('pending')->after('images');
        });
    }
    public function down(): void
    {
        Schema::table('orders', function (Blueprint $table) {
            $table->dropColumn('payment_status');
        });
```

```
    }
};
```

4.2. Реалізований функціонал сайту для користувачів

У процесі наступного етапу практичної реалізації було побудовано універсальну систему навігації на базі Blade-компонентів, що відповідає архітектурній парадигмі Laravel 12. Спочатку в директорії resources/views/components за доктриною фреймворка створено компонент x-layout, який відповідає за фіксовану шапку сайту. Вигляд цього компонента зводиться до наступного фрагмента коду:

```
<header class="fixed max-w-7xl max-h-[1200px] border-l-3 border-r-3 border-violet-400 mx-auto overflow-auto rounded-b-3xl inset-x-0 bg-gradient-to-r from-slate-400 to-stone-400 to-90%">
    <div class="container flex items-center justify-between p-2 pl-16 pr-16">
        <x-logo/>
        <x-nav/>
    </div>
</header>
<div>
    <a href="{{ route('index') }}"
       class="flex items-center text-lg gap-2 font-bold
              hover:text-violet-500 hover:fill-violet-500
              transition-transform duration-300 hover:scale-105">
        <x-icon name="sewing-machine" />
        {{ config('app.name') }}
```

```
    </a>
</div>
```

Застосовано тег <a> з атрибутом href="{{ route('index') }}", який формує посилання на головну сторінку через іменований маршрут index.

Комбінація Tailwind CSS-класів flex items-center gap-2 розташовує іконку і текст у рядок із відступом між ними, а text-lg font-bold задає розмір і жирність шрифту.

Ефекти наведення (hover:) змінюють колір тексту та заливку SVG-іконки на фіолетовий (text-violet-500 / fill-violet-500), а класи transition-transform duration-300 hover:scale-105 забезпечують плавне збільшення масштабу при наведенні курсора.

Компонент <x-icon name="sewing-machine" /> генерує SVG-зображення швейної машинки, а {{ config('app.name') }} підставляє назву додатку з конфігурації.

Таким чином, <x-logo> реалізує інтерактивний логотип, що одночасно є посиланням на головну сторінку та забезпечує єдиний стиль бренду на всіх сторінках сайту.

Згідно з офіційною конвенцією Laravel 12, усі власні Blade-компоненти розміщуються у директорії resources/views/components, а механізм автоматичного розпізнавання імен дає змогу групувати компоненти за папками без необхідності вказувати повні шляхи. У моєму проєкті компонент навігаційного меню було поміщено в каталог sewing-corner/resources/views/components/nav/index.blade.php

Оскільки файл названо index.blade.php всередині папки nav, фреймворк резолвить його під іменем <x-nav>. Такий підхід не лише відповідає внутрішній логіці Laravel, але й спрощує організацію кодової бази, дозволяючи групувати споріднені представлення у відповідні піддиректорії.

Логіку формування пунктів меню реалізовано у компоненті x-nav за допомогою колекції з конфігураційного файлу config/watch.php:

```
@php
    $items = collect(config('watch.nav_items'))->reject(
        fn($label, $routeName) => in_array($routeName, ['terms', 'privacy', 'about'])
    );
@endphp


<nav>
    <ul class="flex gap-2">
        <!-- Пункти меню для гостей -->
        @guest
        @foreach($items->reject( fn($label, $routeName) => in_array($routeName, ['dashboard', 'orders.index' ])) as $routeName => $label)
            <x-nav.item
                :href="route($routeName)"
                :is-active="request()->routeIs($routeName)"
                :class="$routeName === 'login' || $routeName === 'register' ? 'border-2 rounded-lg border-slate-500 px-4 py-2  hover:border-violet-500' : 'px-2 py-1'">
                {{ $label }}
            </x-nav.item>
        @endforeach
        @endguest
```

```
        <!-- Пункти меню для авторизованих користувачів-->
        @auth
        @foreach ($items->reject(    fn($label, $routeName) => in_array($routeName, ['login', 'register'])) as $routeName => $label)
            <x-nav.item
                :href="route($routeName)"
                :is-active="request()->routeIs($routeName)"
                :class="$ routeName === 'dashboard' ? 'border-2 rounded-lg border-slate-500 px-4 py-2 hover:border-violet-500' : 'px-2 py-1'">
                {{ $label }}
            </x-nav.item>
        @endforeach
        @endauth
    </ul>
</nav>
```

У конфігураційному файлі **config/watch.php** визначено масиви для пунктів навігації та соціальних мереж:

```
return [
    'nav_items' => [
        'index'         => 'Головна',
        'about'         => 'Про Нас',
```

```
        'contact'       => 'Контакти',
        'terms'         => 'Умови використання',
        'privacy'       => 'Політика конфіденційності',
        'orders.index'  => 'Замовлення',
        'login'         => 'Вхід',
        'register'      => 'Реєстрація',
        'dashboard'     => 'Мій Кабінет',
    ],
    'social_networks' => [
        'instagram'    => 'https://instagram.com',
        'telegram'     => 'https://telegram.org',
        'whatsapp'     => 'https://whatsapp.com',
        'viber'        => 'https://viber.com',
    ],
];
```

Такий підхід дозволяє централізовано керувати пунктами меню й забезпечує динамічне відображення навігаційних елементів залежно від стану користувача, що сприяє підвищенню гнучкості та підтримуваності коду.

Сторінки з навігаційної панелі

У навігаційній панелі для незареєстрованих користувачів передбачено чотири пункти: «Головна», «Контакти», «Вхід» та «Реєстрація». Після вибору маршруту «Головна» відображається секція, реалізована в файлі hero.blade.php, яка призначена для первинної взаємодії з відвідувачем сайту.

У шаблоні **hero.blade.php** використано директиву @guest, що гарантує показ цього блоку виключно незалогіненим користувачам. Основою служить контейнер із класами Tailwind CSS, які забезпечують флексбокс-верстку в стовпець із центруванням елементів по обох осях і адаптивними відступами. Усередині контейнера виводиться заголовок із шрифтом font-extrabold та розміром text-5xl, що вітально звертається до користувача — «Вітаємо у Sewing Corner». Під ним розміщено підзаголовок із класами text-xl font-medium, який запрошує користувача вибрати подальшу дію.

Далі розташовано два посилання на іменовані маршрути login та register, оформлені як кнопки з фоном bg-violet-400, закругленими кутами та ефектом затемнення при наведені. У разі наявності флеш-повідомлення про успішну операцію (session('success')) під кнопками виводиться відповідний блок із зеленим фоном, рамкою та текстом, що повідомляє про результат попередньої дії. Завдяки поєднанню логіки Blade і утилітарних класів Tailwind CSS забезпечується інтуїтивна навігація та єдність стилю на початковій сторінці веб-застосунку:

```
<section>
  @guest
  <div class="container flex flex-col gap-6 justify-center items-center min-h-screen py-40 px-4">
    <div>
      <h1 class="font-extrabold text-5xl">Вітаєммо у Sewing Corner</h1>
    </div>
    <p class="text-xl font-medium">Що обирете?</p>
    <div class="flex gap-2 mt-6">
```

```
<a href="{{ route('login') }}" class="whitespace-nowrap font-semibold px-14 py-2 bg-violet-400 hover:bg-violet-500 rounded-xl flex justify-center items-center w-40">Вийти</a>

<a href="{{ route('register') }}" class="whitespace-nowrap font-semibold px-14 py-2 bg-violet-400 hover:bg-violet-500 rounded-xl flex justify-center items-center w-40">Зареєструватися</a>

</div>

@if (session('success'))

<div class="mb-4 p-4 bg-green-100 border border-green-400 text-green-700 rounded">

{{ session('success') }}

</div>

@endif

</div>

@endguest

</section>
```

Сторінка Контакти

На сторінці «Контакти» (Рис. 4.2.1) застосовано загальний макет x-layout, що забезпечує цілісність структури та стилю інтерфейсу, реалізована як нащадок базового макета через компонент:

```
<x-layout>

<div class="flex flex-col items-center justify-center min-h-screen gap-16">
```

```
{{-- Телефон --}}
<div class="flex flex-col items-center gap-3 text-center">
  <p class="text-violet-600 text-2xl font-semibold tracking-wide">Номер телефону:</p>
  <p class="select-all cursor-pointer text-4xl font-bold text-slate-800 hover:text-violet-700 transition duration-300">
    +3809911223344
  </p>
</div>

{{-- Соцмережі --}}
<div class="flex flex-col items-center gap-4 text-center">
    <p class="text-violet-600 text-2xl font-semibold tracking-wide">Соціальні мережі:</p>
  <ul class="flex gap-6 flex-wrap justify-center">
    @foreach (config('watch.social_networks') as $name => $href)
      <li>
        <a href="{{ $href }}" target="_blank" rel="noopener noreferrer">
          <x-icon :$name class="size-16 fill-slate-700 hover:fill-violet-600 transition duration-300 transform hover:scale-110" />
        </a>
      </li>
```

```
            @endforeach
        </ul>
    </div>
</div>
</x-layout>
```

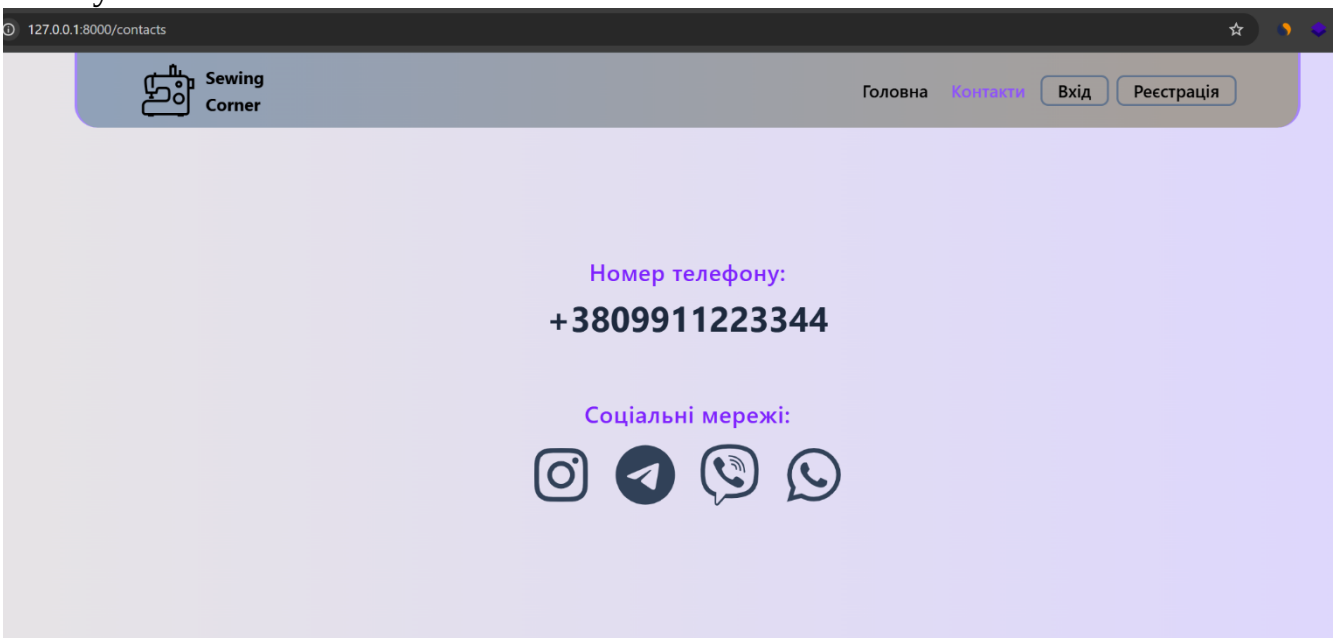

Рисунок 4.2.1 Сторінка Контакти

У цьому шаблоні контейнер із класами Tailwind CSS (flex flex-col items-center justify-center min-h-screen gap-16) забезпечує центральне вирівнювання та адаптивні відступи. Перший блок демонструє контактний номер із стилями для виділення тексту та ефектами при наведенні, що підвищує зручність копіювання. Друга секція формує список зовнішніх посилань на соціальні мережі: масив config('watch.social_networks') читається з конфігураційного файлу config/watch.php, де ключі відповідають іменам SVG-іконок, а значення — URL-адресам. Для кожного елемента масиву генерується тег <a> з атрибутами target="_blank" і rel="noopener noreferrer", що гарантують безпечне відкриття зовнішніх ресурсів у новій вкладці. Усередині посилання викликається компонент <x-icon>, розташований у файлі sewing-corner/resources/views/components/icon.blade.php (Рис. 4.2.2) який повертає SVG-

зображення відповідного імені. Завдяки використанню SVG забезпечується висока якість і масштабованість піктограм незалежно від роздільності екрана. Поєднання централізованого збереження посилань у конфігурації, динамічного рендерингу через Blade і утилітарного підходу Tailwind CSS дозволяє підтримувати єдиний стиль та оперативно редагувати контактну інформацію без зміни шаблонів.

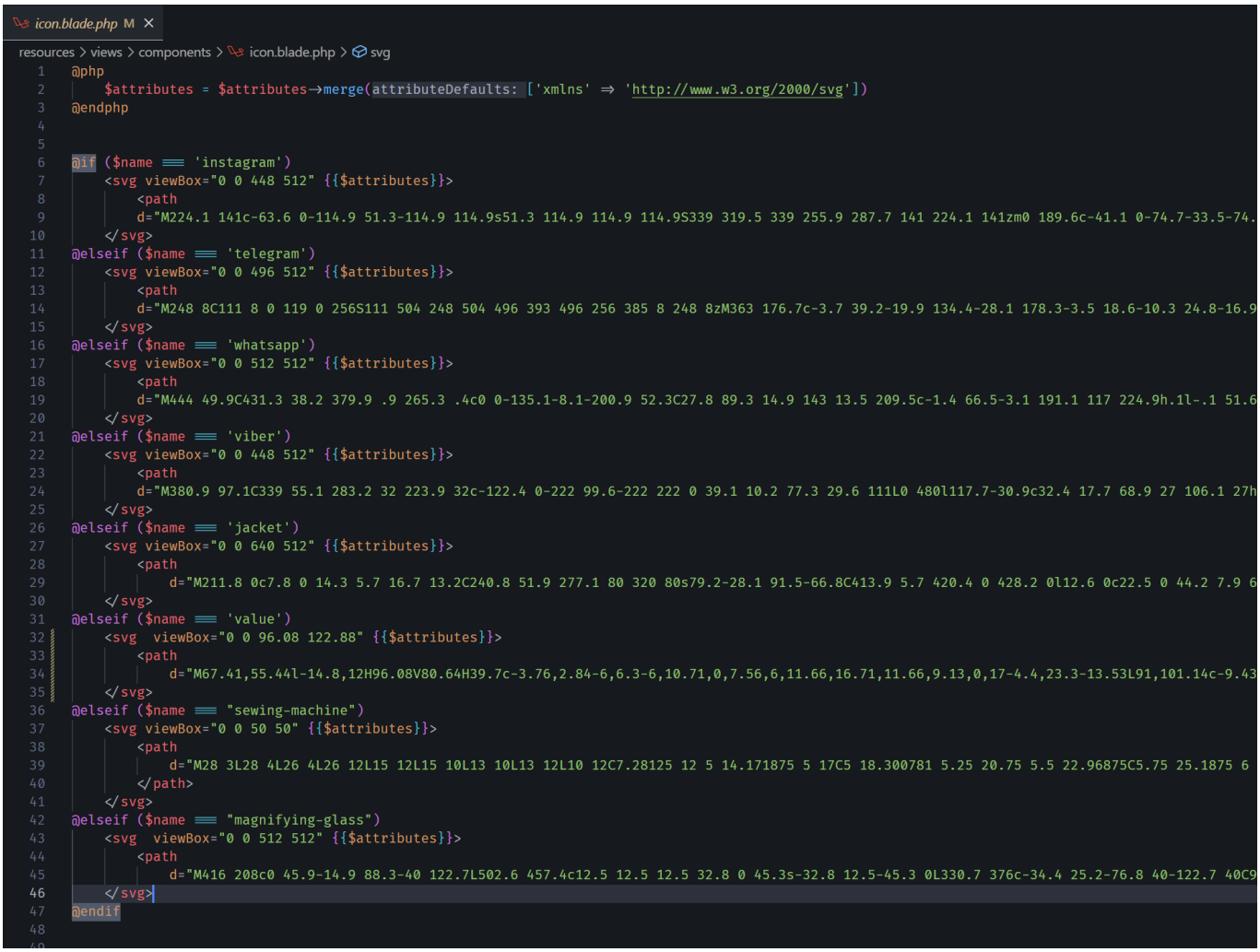

Рисунок 4.2.2 icon.blade.php

Сторінка Входу

Сторінка авторизації login.blade.php реалізована як нащадок загального макету x-layout, що забезпечує єдину структуру й стилістику всього інтерфейсу. Основний вміст сторінки поміщено в контейнер із класами Tailwind CSS, які централізують форму

авторизації на всьому екрані (flex items-center justify-center min-h-screen) та задають фон із плавним градієнтом, межу й тінь для візуального виділення блоку:

```
<div class="w-full max-w-md p-8 bg-gradient-to-br from-slate-300 to-stone-300
        border-2 border-slate-400 rounded-2xl shadow-2xl">
```

У верхній частині форми розташовано заголовок із класами text-3xl font-bold text-center, який інформує користувача про призначення сторінки — «Вхід». Перед полями введення за допомогою директив Blade виводяться флеш-повідомлення про помилки валідації та повідомлення про успішні попередні дії:

```
@if ($errors->any())
    <div class="mb-4 text-sm text-red-700 bg-red-100 border border-red-300 px-4 py-3 rounded-lg">
        <ul class="list-disc pl-5">
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif


@if (session('success'))
    <div class="mb-4 bg-green-100 border border-green-400 text-green-700 px-4 py-3 rounded-lg text-sm">
```

```
        {{ session('success') }}
    </div>
@endif
```

Форма авторизації містить поля для введення імені користувача або електронної пошти та пароля, кожне з яких оформлено утилітарними класами Tailwind для забезпечення адаптивності, контрастності та інтерактивності (фокусні стани focus:ring-violet-500 focus:border-violet-500). Під полями передбачено чекбокс «Запам’ятати мене» для встановлення довготривалої сесії. Кнопка відправки форми поєднує класи bg-violet-600 text-white rounded-lg shadow з анімаційним збільшенням масштабу при наведенні (hover:scale-105 duration-200), що підсилює користувацький досвід.

```
<form method="POST" action="{{ route('login') }}" class="space-y-5">
    @csrf
    <!-- Поля login і password, checkbox remember -->
    <div class="flex justify-center">
        <button type="submit" class="w-full py-3 bg-violet-600 text-white ... hover:scale-105">
            Увійти
        </button>
    </div>
    <div class="text-center mt-4">
        <a href="{{ route('password.request') }}" class="text-sm text-gray-600 hover:text-violet-500 hover:underline">
            Забули пароль?
        </a>
```

```
    </div>
</form>
```

Таким чином, вигляд сторінки входу (Рис. 4.2.3).

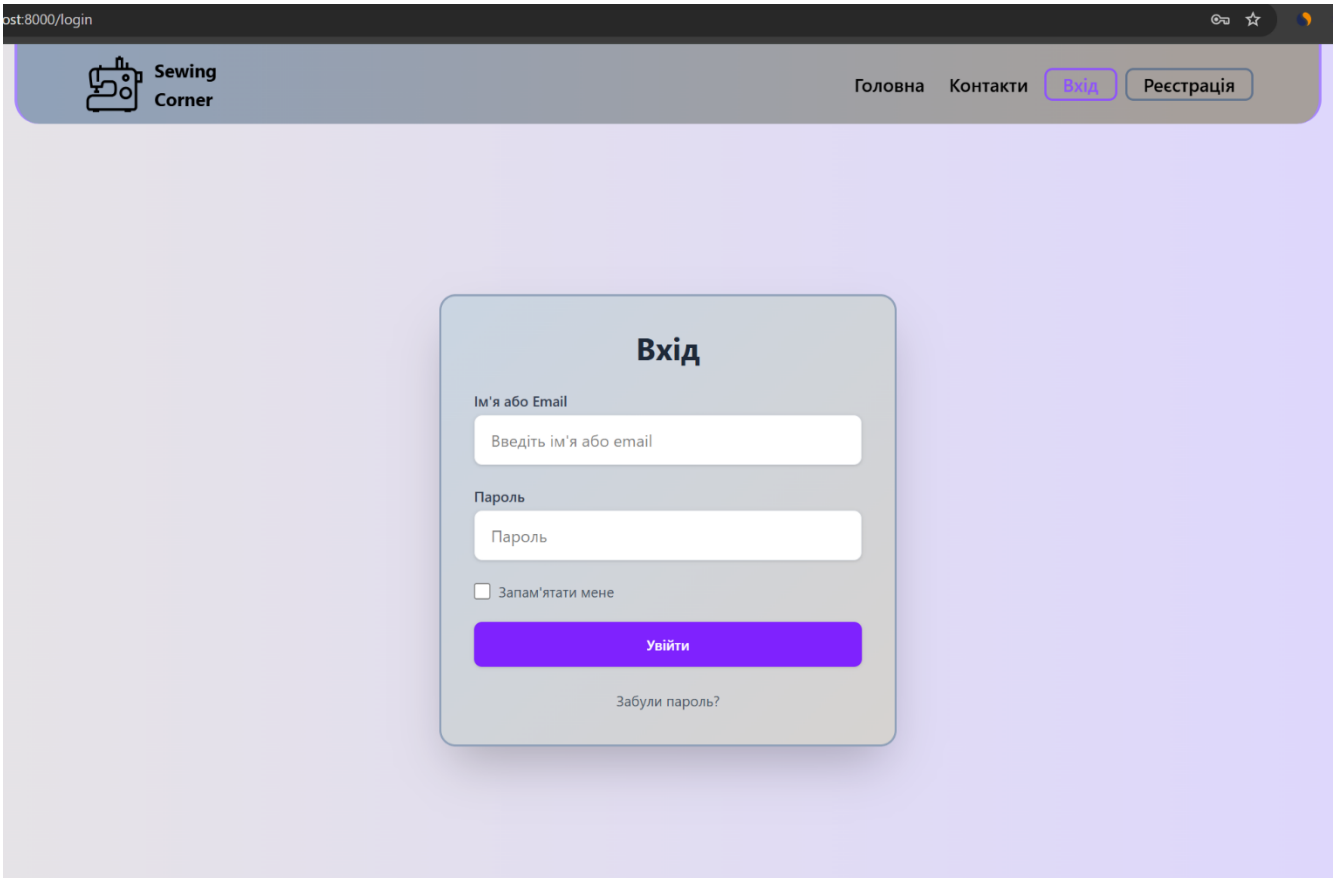

Рисунок 4.2.3 -Сторінка входу

Сторінка Реєстрації

Шаблон register.blade.php (Рис. 4.2.4) реалізовано як спадкоємець базового макета x-layout із метою забезпечення єдиної структури та стилю інтерфейсу. Головний контейнер сторінки вирівнюється по центру екрану за допомогою утиліт Tailwind CSS (flex items-center justify-center min-h-screen), а внутрішній блок з формою має обмеження ширини (w-full max-w-md), паддінги (p-8), градієнтне тло (bg-gradient-to-br from-slate-

300 to-stone-300), окантовку та тінь (border-2 border-slate-400 rounded-2xl shadow-2xl), що підкреслює візуальну автономність секції реєстрації.

Заголовок форми задається тегом <h2> з класами text-2xl font-bold mb-4 text-center text-gray-800, який інформує користувача про призначення сторінки. Безпосередньо під ним Blade-директива @if ($errors->any()) перевіряє наявність помилок валідації і при їх виявленні виводить HTML-блок з червоним текстом (text-red-600), у якому циклом @foreach ($errors->all() as $error) перераховуються повідомлення про некоректне заповнення полів. Це рішення забезпечує керовану обробку помилок і підвищує зручність взаємодії з формою.

Форма визначена з атрибутами method="POST" та action="{{ route('register') }}", а директива @csrf автоматично додає захисний токен проти CSRF-атак. Поля введення—ім’я, електронна пошта, телефон, пароль та його підтвердження—створені як стандартні <input> з типами text, email і password відповідно, із застосуванням утилітарних класів Tailwind для фокусних та валідаційних станів (focus:ring-violet-500 focus:border-violet-500). Значення полів текстового вводу попередньо заповнюються за допомогою хелпера old(), що дозволяє зберігати введені дані після помилки валідації. Кнопка відправки реалізована через <button type="submit"> із поєднанням класів bg-violet-600 text-white rounded-lg shadow hover:bg-violet-500 transition transform hover:scale-105 duration-200, що забезпечує послідовну стилістику й інтерактивний ефект наведення.

У нижній частині форми передбачено посилання на маршрут логіну (Увійти), що полегшує навігацію між режимами аутентифікації. Така організація шаблону дозволяє реалізувати функціонал реєстрації користувача в межах парадигми MVC — контролер обробляє запит, валідує дані та повертає назад цей Blade-шаблон з відповідними параметрами, а представлення відповідає за вивід форми

й повідомлень у зручному для користувача вигляді.

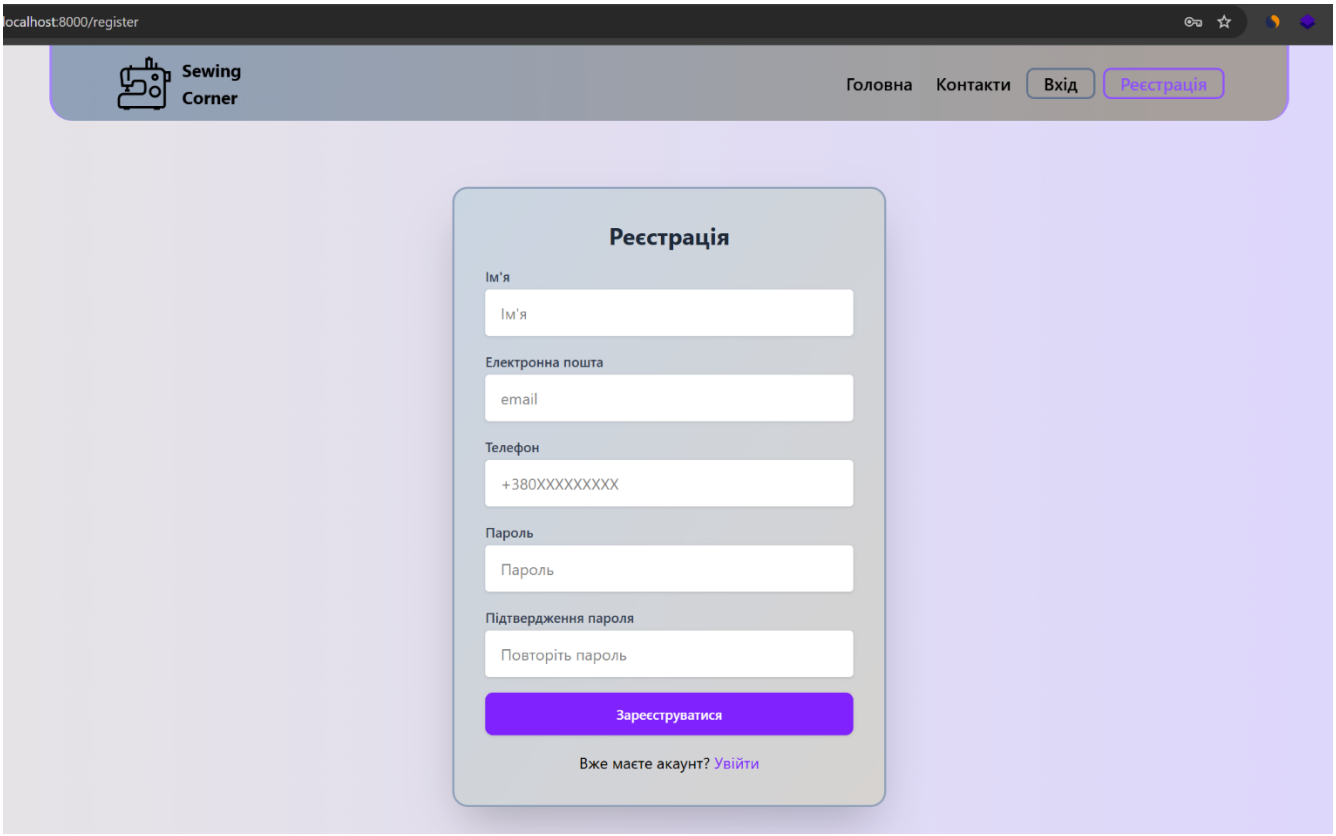

Рисунок 4.2.4 Сторінка реєстрації

Footer

У файлі resources/views/components/footer/index.blade.php визначено основний контейнер футера, який забезпечує візуальну обрамленість та структуру сторінки:

```
<footer class="border-l-3 border-r-3 border-violet-400 rounded-t-3xl bg-gradient-to-r from-stone-400 to-slate-400 to-90%">
    <div class="flex flex-col gap-8 container py-4 px-4 pr-12 pl-12">
        <div class="flex items-center gap-8 justify-between">
            <x-footer.nav/>
            <x-footer.social/>
```

```
        </div>
        <div class="text-center">
          <x-footer.copyright/>
        </div>
    </div>
</footer>
```

У цьому компоненті застосовані класи Tailwind CSS для визначення товщини лівої і правої рамок (border-l-3, border-r-3), закругленого верхнього краю (rounded-t-3xl) та градієнтного фону. Внутрішня структура організована у дві горизонтальні секції: перша містить навігаційні блоки, друга—авторське право.

Компонент навігації resources/views/components/footer/nav/index.blade.php динамічно формує список пунктів, виключаючи маршрути, які недоречні у футері:

```
@php
    $items = collect(config('watch.nav_items'))->reject(
        fn($label, $routeName) => in_array($routeName, ['index', 'login', 'register', 'orders.index', 'profile', 'dashboard'])
    );
@endphp

<ul class="flex gap-4">
    @foreach ($items as $routeName => $label)
        <x-footer.nav-item :href="route($routeName)">{{ $label }}</x-footer.nav-item>
```

```
    @endforeach
</ul>
```

За допомогою функцій Laravel Collection виконується фільтрація масиву nav_items з файлу config/watch.php, після чого кожен елемент передається у компонент <x-footer.nav-item>.

Сам компонент resources/views/components/footer/nav/nav-item.blade.php відповідає за представлення одного пункту меню:

```
<li>
    <a class="font-medium hover:text-purple-200" {{ $attributes }}>
        {{ $slot }}
    </a>
</li>
```

Тут атрибути посилання ({{ $attributes }}) підтягують маршрут із дочірнього шаблону, а $slot виводить текст пункту. Стилі гарантують середню жирність шрифту та зміну кольору при наведенні.

Блок соціальних мереж реалізований у файлі resources/views/components/footer/social.blade.php і використовує конфігураційний масив social_networks:

```
<ul class="flex gap-4">
    @foreach (config('watch.social_networks') as $name => $href)
        <li>
            <a href="{{ $href }}">
                <x-icon :$name class="size-9 fill-slate-800 hover:fill-violet-200"/>
```

```
            </a>
        </li>
    @endforeach
</ul>
```

Кожне посилання містить компонент <x-icon>, який повертає SVG-іконку за іменем мережі. Використання SVG забезпечує високу якість і масштабованість.

Таким чином, footer (Рис. 4.2.5) складається з трьох окремих, чітко інкапсульованих компонентів, що відповідає принципам модульності та повторного використання коду в архітектурі Laravel. Це полегшує підтримку та розширення функціоналу футера без зміни інших частин представлення.

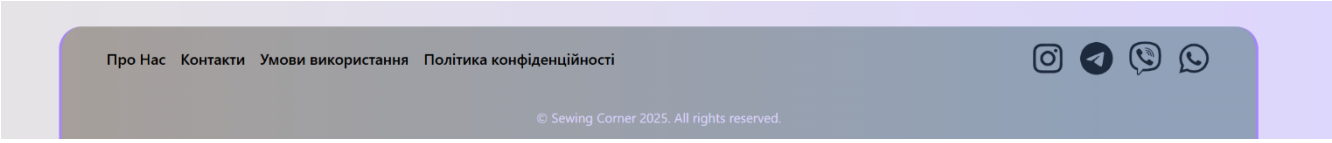

Рисунок 4.2.5 Footer

Опис сторінок в Footer

Сторінка «Про нас» реалізована як простий статичний шаблон у межах компонента x-layout. Основний текст розташовано в центрі екрану за допомогою утиліт Tailwind CSS (flex items-center justify-center min-h-screen), а всередині — блок із класами font-bold text-7xl text-center, що задають жирний великий шрифт і центроване вирівнювання:

```
<x-layout>
    <div class="flex items-center justify-center min-h-screen">
        <div class="flex flex-col font-bold text-7xl">
            Про нас
        </div>
```

```
        </div>
</x-layout>
```

Сторінка «Контакти» (resources/views/pages/contact.blade.php) дублює розділ контактної інформації, реалізований у футері, і доступна за маршрутом /contacts. Вона наслідує базовий макет через компонент <x-layout> і виводить ті ж дані — номер телефону та перелік соціальних мереж із конфігурації — що й у футері. Таким чином, реалізовано принцип DRY: одна й та сама логіка відображення контактів централізовано визначається в окремому шаблоні, а футер лише містить посилання на цю сторінку без дублювання контенту.

Сторінка «Умови використання» (Рис. 4.2.6) реалізована як спадкоємець макета x-layout із централізованим вирівнюванням вмісту за допомогою утиліт Tailwind CSS (flex items-center justify-center min-h-screen). Основний блок містить заголовок, виконаний у вигляді жирного тексту великого розміру (font-bold text-7xl), що відображає текст «Privacy»:

```
<x-layout>
    <div class="flex items-center justify-center min-h-screen">
        <div class="flex flex-col font-bold text-7xl">
            Privacy
        </div>
    </div>
</x-layout>
```

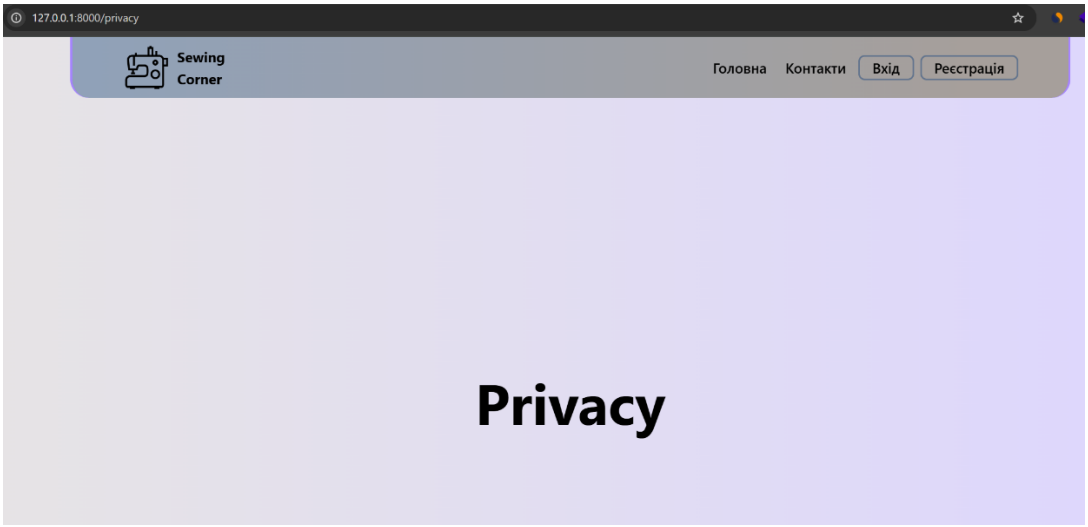

Рисунок 4.2.6 Сторінка умови використання

Сторінка Політика конфіденційності:

Сторінка «Політика конфіденційності» реалізована аналогічно іншим статичним розділам і наслідує базовий макет через компонент <x-layout>:

```
<x-layout>
    <div class="flex items-center justify-center min-h-screen">
        <div class="flex flex-col font-bold text-7xl">
            Terms
        </div>
    </div>
</x-layout>
```

Посилання на соц мережі:

Посилання на соціальні мережі (Рис. 4.2.7) формується через перебір конфігураційного масиву config('watch.social_networks') у sewing-corner\config\watch.php. Для кожного запису створюється елемент списку <li> з

гіперпосиланням на відповідний URL, а вмістом є SVG-іконка через компонент <x-icon>. Використання SVG забезпечує масштабованість, а утилітарні класи Tailwind (flex gap-4, size-9, hover:fill-violet-200) гарантують єдиний стиль**:**

```
<ul class="flex gap-4">
        @foreach (config('watch.social_networks') as $name => $href)
        <li>
           <a href="{{ $href }}">
              <x-icon :$name class="size-9 fill-slate-800 hover:fill-violet-200"/>
           </a>
        </li>
        @endforeach
</ul>
```

Рисунок 4.2.7 Посилання на соціальні мережі у footer

Кабінет Користувача

У шаблоні dashboard-layout.blade.php реалізовано базову структуру вітальної сторінки користувача з адаптивним інтерфейсом. У центрі відображається персоналізоване привітання користувача на основі автентифікаційних даних (auth()->user()->name). Шаблон також містить розділи @yield('subtitle') та @yield('content'), що дає змогу динамічно підставляти підзаголовки й контент залежно від контексту

використання:

```
<x-layout>
    <div class="flex flex-col items-center justify-center w-full max-w-6xl mx-auto p-8 rounded-xl shadow-xl border-2 border-slate-400 bg-gradient-to-l from-slate-400 to-stone-300 mt-32">
        {{-- Заголовок --}}
        <h1 class="text-4xl font-bold text-center">
            Вітаємо, <span class="text-violet-500">{{ auth()->user()->name }}</span>!
        </h1>
        <p class="text-lg text-gray-600 mt-4 text-center">
            @yield('subtitle') {{-- Динамічний підзаголовок --}}
        </p>
        <div class="mt-10 w-full">
            @yield('content') {{-- Динамічний контент --}}
        </div>
    </div>
</x-layout>
```

Шаблон Blade **dashboard.blade.php** реалізує інтерфейс особистого кабінету користувача у веб-додатку (Рис.4.2.10), розробленому з використанням Laravel. Структура компонента базується на головному макеті <x-layout> і є стилізованою за допомогою утиліт Tailwind CSS. Основне призначення цього шаблону — забезпечення інтерактивного доступу до основних функцій, пов’язаних із керуванням користувацьким профілем та замовленнями:

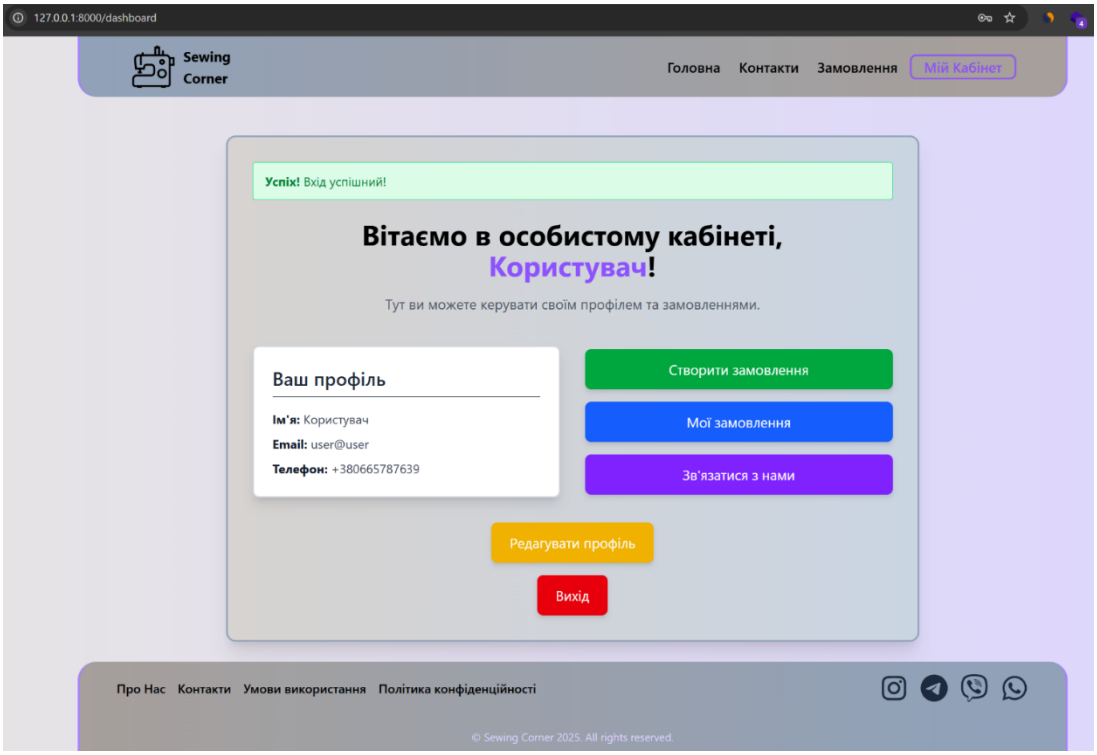

Рисунок 4.2.10 – Кабінет користувача

```
<x-layout>

    <div class="flex flex-col items-center justify-center w-full max-w-4xl mx-auto p-8 rounded-xl shadow-xl border-2 border-slate-400 bg-gradient-to-l from-slate-300 to-stone-300 mt-32">

        {{-- Повідомлення про успіх --}}

        @if (session('success'))

            <div class="bg-green-100 border border-green-400 text-green-700 px-4 py-3 rounded relative mb-6 w-full" role="alert">

                <strong class="font-bold">Успіх!</strong>

                <span class="block sm:inline">{{ session('success') }}</span>

            </div>

        @endif
```

```
<h1 class="text-4xl font-bold text-center">
    Вітаємо в особистому кабінеті,<br>
    <span class="text-violet-500">{{ auth()->user()->name }}</span>!
</h1>
<p class="text-lg text-gray-600 mt-4 text-center">
    Тут ви можете керувати своїм профілем та замовленнями.
</p>
{{-- Блоки профілю та дій --}}
<div class="grid grid-cols-1 md:grid-cols-2 gap-8 mt-10 w-full">
    {{-- Інформація про профіль --}}
    <div class="border-2 border-gray-300 bg-white shadow-lg rounded-lg p-6">
        <h2 class="text-2xl font-semibold text-gray-800 mb-4 border-b pb-2">          Ваш профіль</h2>
        <ul class="text-gray-700 space-y-2">
            <li><strong class="text-gray-900">  Ім'я:</strong> {{ auth()->user()->name }}</li>
            <li><strong class="text-gray-900">Email:</strong> {{ auth()->user()->email }}</li>
            <li><strongclass="text-gray-900"> Телефон:</strong> {{ auth()->user()->phone ?? 'Не вказано' }}</li>
        </ul>
    </div>
    {{-- Кнопки дій --}}
```

```
        <div class="flex flex-col justify-center items-center space-y-4">
            <a href="{{ route('create-order') }}"
               class="px-6 py-3 bg-green-600 text-white rounded-lg shadow-md hover:bg-green-500 focus:ring-2 focus:ring-green-500 focus:ring-offset-2 inline-block text-center transition duration-300 ease-in-out transform hover:scale-105 w-full text-lg">
                Створити замовлення
            </a>
            <a href="{{ route('orders.index') }}"
               class="px-6 py-3 bg-blue-600 text-white rounded-lg shadow-md hover:bg-blue-500 focus:ring-2 focus:ring-blue-500 focus:ring-offset-2 inline-block text-center transition duration-300 ease-in-out transform hover:scale-105 w-full text-lg">
                Мої замовлення
            </a>
            <a href="{{ route('contact') }}"
               class="px-6 py-3 bg-violet-600 text-white rounded-lg shadow-md hover:bg-violet-500 focus:ring-2 focus:ring-violet-500 focus:ring-offset-2 inline-block text-center transition duration-300 ease-in-out transform hover:scale-105 w-full text-lg">
                Зв'язатися з нами
            </a>
        </div>
    </div>
    {{-- Кнопка редагування профілю та виходу  --}}
```

```
        <div class="flex flex-col justify-center items-center gap-4 mt-8 w-full">
            <a href="{{ route('profile.edit') }}"
               class="px-6 py-3 bg-yellow-500 text-white rounded-lg shadow-md hover:bg-yellow-400 focus:ring-2 focus:ring-yellow-400 focus:ring-offset-2 transition transform hover:scale-105 duration-300 text-lg text-center">
                Редагувати профіль
            </a>
            <form id="logout-form" action="{{ route('logout') }}" method="POST">
                @csrf
                <button
                    type="submit"
                    class="px-6 py-3 bg-red-600 text-white hover:bg-red-500 rounded-lg shadow-md cursor-pointer focus:ring-2 focus:ring-red-500 focus:ring-offset-2 transition duration-300 ease-in-out transform hover:scale-105 text-lg text-center"
                >
                    Вихід
                </button>
            </form>
        </div>
    </div>
</x-layout>
```

Ця розміітка створює сторінку особистого кабінету користувача в межах загального шаблону. Спочатку, якщо у сесії є повідомлення про успіх, воно відображається зверху у вигляді зеленого алерта. Далі виводиться привітання з ім’ям поточного користувача, а також короткий опис, що ця сторінка дозволяє керувати профілем та замовленнями.

Основна частина поділена на дві колонки: зліва показується інформація про користувача (ім’я, email, телефон), а справа — кнопки для створення нового замовлення, перегляду своїх замовлень і звернення до служби підтримки. Внизу розміщені кнопки для редагування профілю та виходу з акаунту. Усі елементи стилізовані за допомогою Tailwind CSS для гарного вигляду і зручності використання.

Сторінка Редагування профілю

Даний файл edit-profile.blade.php реалізує інтерфейс користувача для редагування персональних даних у веб-застосунку. Після активації кнопки "Редагувати профіль" в особистому кабінеті користувач перенаправляється на відповідну сторінку, де представлено форму для зміни основної інформації: імені, електронної пошти та номера телефону.

Форма (Рис.4.2.11) містить поля, значення яких ініціалізуються поточними даними користувача, отриманими із серверної частини, або попередніми введеними значеннями у разі помилки валідації. Надсилання форми здійснюється за HTTP методом PUT на визначений маршрут profile.update, що передбачає оновлення відповідних записів у базі даних.

Інтерфейс побудований із застосуванням фреймворку Tailwind CSS, що забезпечує сучасний дизайн та інтерактивність елементів керування, таких як кнопки з анімацією при наведенні. Також на сторінці передбачено посилання для повернення до особистого кабінету, що сприяє покращенню навігації користувача в системі.

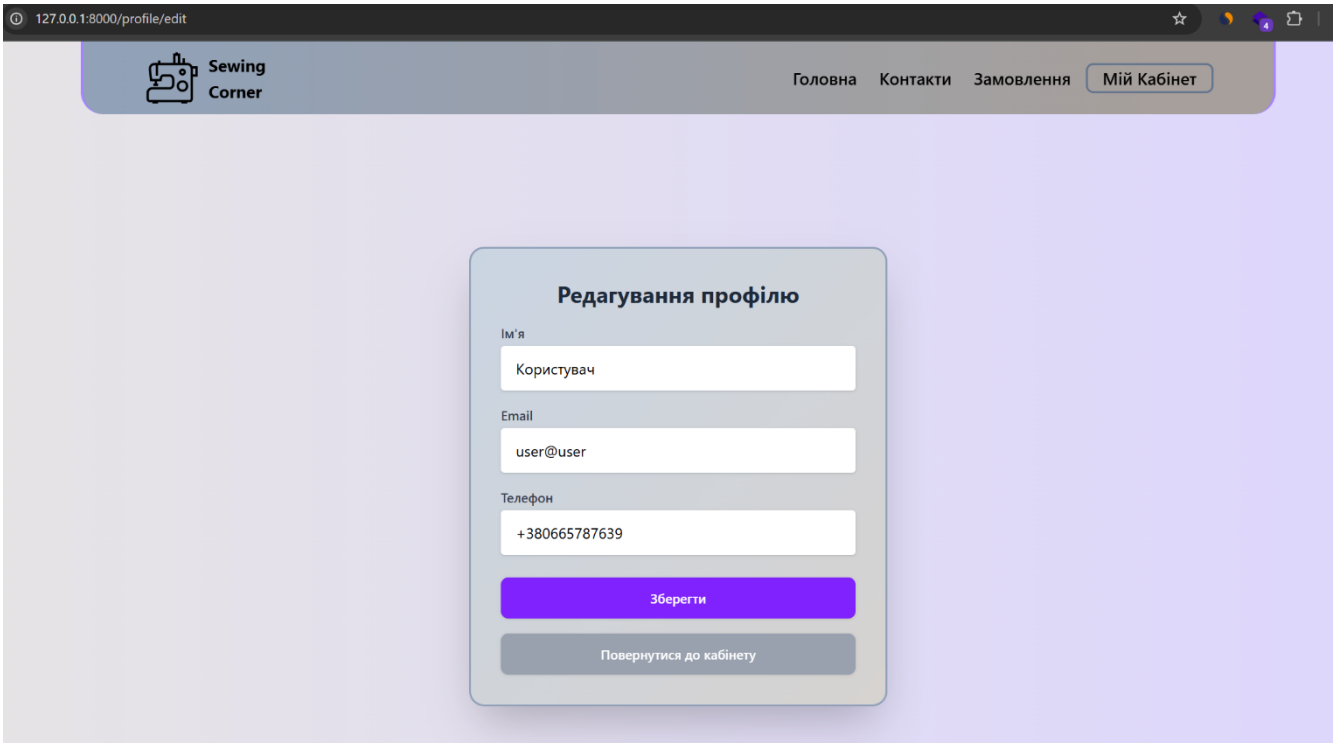

Рисунок 4.2.11 Сторінка редагування профілю

Сторінка Створення замовлення

Файл create.blade.php реалізує форму для створення нового замовлення на сайті (Рис. 4.2.12). Після натискання кнопки "Створити замовлення" в особистому кабінеті користувач переходить на цю сторінку.

Інтерфейс містить форму з кількома обов’язковими полями: вибір типу послуги (ремонт, пошив або інше), текстове поле для детального опису роботи, а також завантаження двох зображень, які ілюструють замовлення. Поле для вибору файлів приховане, замість нього користувач використовує стилізовану кнопку, що відкриває діалог вибору файлів.

Валідація на стороні клієнта обмежує завантаження не більше двох зображень. Після вибору файлів у відповідних елементах відображаються прев’ю зображень для попереднього перегляду. Якщо користувач вибирає більше двох файлів, система повідомляє про помилку та очищує вибір.

При відправленні форма надсилається методом POST на маршрут store-order, де здійснюється обробка і збереження даних замовлення. Візуальне оформлення сторінки виконане з використанням Tailwind CSS, що забезпечує сучасний та зручний для користувача інтерфейс із плавними анімаціями і зворотнім зв’язком.

Розмітка сторінки:

```
<x-layout>
  <div class="flex items-center justify-center min-h-screen">
    <div class="max-w-2xl w-full p-8 rounded-xl shadow-xl border-2 border-slate-400 bg-gradient-to-l from-slate-400 to-stone-300">
      <h2 class="text-2xl font-bold mb-6 text-center">Створення замовлення</h2>

      {{-- Перевірка на наявність помилок --}}
      @if ($errors->any())
        <div class="mb-4 text-red-600">
          <ul>
            @foreach ($errors->all() as $error)
              <li>{{ $error }}</li>
            @endforeach
          </ul>
        </div>
      @endif
```

```
<form method="POST" action="{{ route('store-order') }}" enctype="multipart/form-data">
    @csrf

    <div class="mb-6">
        <label for="service_type" class="block text-sm font-medium text-gray-700 mb-2">Тип послуги</label>
        <select id="service_type" name="service_type" class="block w-full border border-gray-300 bg-white rounded-lg shadow-sm focus:ring-violet-500 focus:border-violet-500 px-4 py-2">
            <option value="repair">Ремонт</option>
            <option value="sewing">Пошив</option>
            <option value="other">Інше</option>
        </select>
    </div>

    <div class="mb-6">
        <label for="description" class="block text-sm font-medium text-gray-700 mb-2">Опис роботи</label>
        <textarea id="description" name="description" rows="4" class="mt-1 block w-full border border-gray-300 bg-white rounded-lg shadow-sm focus:ring-violet-500 focus:border-violet-500 px-4 py-2 resize-none" placeholder="Додайте опис роботи..."></textarea>
    </div>
```

```
<div class="flex flex-col mb-6">
    <label class="block text-sm font-medium text-gray-700 mb-2">Додати зображення (обов'язково 2)</label>
    <input type="file" id="image_upload" name="images[]" accept="image/*" multiple class="hidden">
    <label for="image_upload" class="cursor-pointer px-4 py-2 bg-gray-300 rounded-lg shadow hover:bg-gray-400 text-center">Обрати файли</label>
    <div id="preview" class="grid grid-cols-2 gap-4 mt-4">
      < img id="preview1" class="w-full h-40 object-cover rounded-lg shadow-md hidden">
      < img id="preview2" class="w-full h-40 object-cover rounded-lg shadow-md hidden">
    </div>
  </div>

  <div class="flex justify-center mt-6 gap-4">
    <button type="submit" class="px-4 py-2 bg-violet-600 text-white rounded shadow hover:bg-violet-500 focus:ring-2 focus:ring-violet-500 focus:ring-offset-2 cursor-pointer transition duration-300">
      Створити замовлення
    </button>
  </div>
```

```
            <div class="flex justify-center mt-2">
                <a href="{{ url('/orders') }}" class="px-4 py-2 bg-red-600 text-gray-800 rounded shadow hover:bg-red-500 focus:ring-2 focus:ring-gray-400 focus:ring-offset-2 cursor-pointer transition duration-300">
                    Відмінити
                </a>
            </div>
        </form>
    </div>
</div>
</x-layout>

<script>
document.getElementById('image_upload').addEventListener('change', function(event) {
    const files = event.target.files;

    if (files.length > 2) {
        alert('Можна завантажити тільки 2 зображення!');
        event.target.value = ''; // Очистити вибір зображень
        return;
    }
```

```
    const preview1 = document.getElementById('preview1');
    const preview2 = document.getElementById('preview2');

    Array.from(files).forEach((file, index) => {
        const reader = new FileReader();
        reader.onload = function(e) {
            if (index === 0) {
                preview1.src = e.target.result;
                preview1.classList.remove('hidden');
            } else if (index === 1) {
                preview2.src = e.target.result;
                preview2.classList.remove('hidden');
            }
        };
        reader.readAsDataURL(file);
    });
});
</script>
```

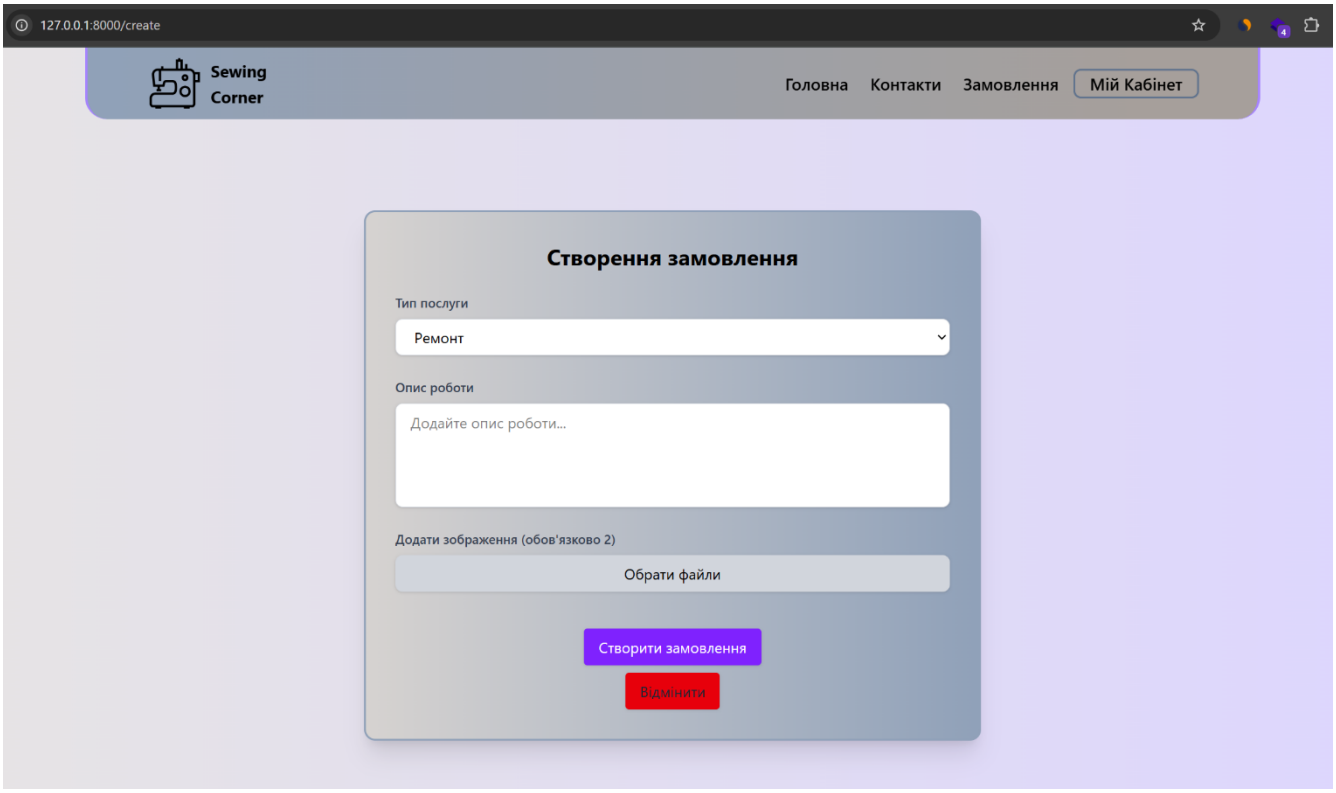

Рисунок 4.2.12 Сторінка створення замовлення

Сторінка замовлення

Сторінка index.blade.php, що розміщується за resources\views\pages\orders\index.blade.php, відповідає за відображення замовлень (Рис. 4.2.13), які пов’язані з автентифікованим користувачем. Вона відкривається після переходу з особистого кабінету через натискання кнопки «Мої замовлення».

Інтерфейс реалізовано у вигляді розділу <section> із зовнішніми відступами (pt-20 px-10) та внутрішньою структурою у вигляді вертикального флекса, що дозволяє впорядковано відображати всі елементи.

На початку сторінки відображається заголовок. Його вміст динамічно визначається залежно від ролі користувача: якщо автентифікований користувач є адміністратором (тобто його роль \App\Enums\UserRole::Admin), тоді заголовком буде «Замовлення користувачів». У протилежному випадку, коли користувач є звичайним клієнтом,

заголовок звучатиме як «Ваші Замовлення». Це дозволяє реалізувати загальну сторінку перегляду замовлень із відмінностями у логіці виведення лише через одну перевірку.

Після заголовка розташовано кнопку створення нового замовлення. Вона оформлена у стилі Tailwind CSS з адаптивною поведінкою — змінює колір і масштаб при наведенні. Кнопка веде на маршрут create-order, який відповідає за перехід на сторінку з формою створення нового замовлення.

Далі на сторінці підключається компонент Blade — <x-order-search/>. Цей компонент, імовірно, відповідає за реалізацію інтерфейсу пошуку або фільтрації замовлень за заданими критеріями. Його точна логіка не представлена у цьому фрагменті, але він інтегрується в загальний потік сторінки, сприяючи покращенню зручності використання, особливо для адміністраторів із великою кількістю замовлень.

Основну частину вмісту становить сітка замовлень, реалізована за допомогою класу grid grid-cols-2 gap-4, що забезпечує двоколонкове відображення з однаковими проміжками. Для кожного замовлення в циклі @foreach викликається Blade-компонент <x-order>, який приймає конкретний екземпляр моделі $order. Це дозволяє уніфіковано відображати всі замовлення відповідно до дизайну, закладеного в компоненті.

Наприкінці сторінки перевіряється наявність методу links() у колекції $orders. Якщо такий метод існує, Laravel автоматично генерує елементи пагінації. Це дозволяє поділити великий список замовлень на окремі сторінки й реалізувати зручну навігацію.

Загалом сторінка index.blade.php є функціональною частиною особистого кабінету, яка об’єднує логіку перегляду, створення та пошуку замовлень, адаптуючись до ролі користувача:

```
<x-layout>

    <section class="pt-20 px-10">
        <div class="container mx-auto flex flex-col gap-6 py-6">
            @if (auth()->user()->role === \App\Enums\UserRole::Admin)
```

```
            <h2 class="text-center font-bold text-5xl">Замовлення користувачів</h2>
        @else
            <h2 class="text-center font-bold text-5xl">Ваші Замовлення</h2>
        @endif
        <div class="w-full mx-auto flex justify-center items-center">
            <ahref="{{ route('create-order') }}" class="w-1/2 px-6 py-3 bg-violet-600 text-white rounded-lg shadow-md hover:bg-violet-500 focus:ring-2 focus:ring-violet-500 focus:ring-offset-2 inline-block text-center transition duration-300 ease-in-out transform hover:scale-105">
                Створити замовлення
            </a>
        </div>
        <x-order-search/>
        <div class="grid grid-cols-2 gap-4">
            @foreach ($orders as $order)
                <x-order :$order/>
            @endforeach
        </div>
        @if(method_exists($orders, 'links'))
            {{ $orders->links() }}
        @endif
    </div>
  </section>
</x-layout>
```

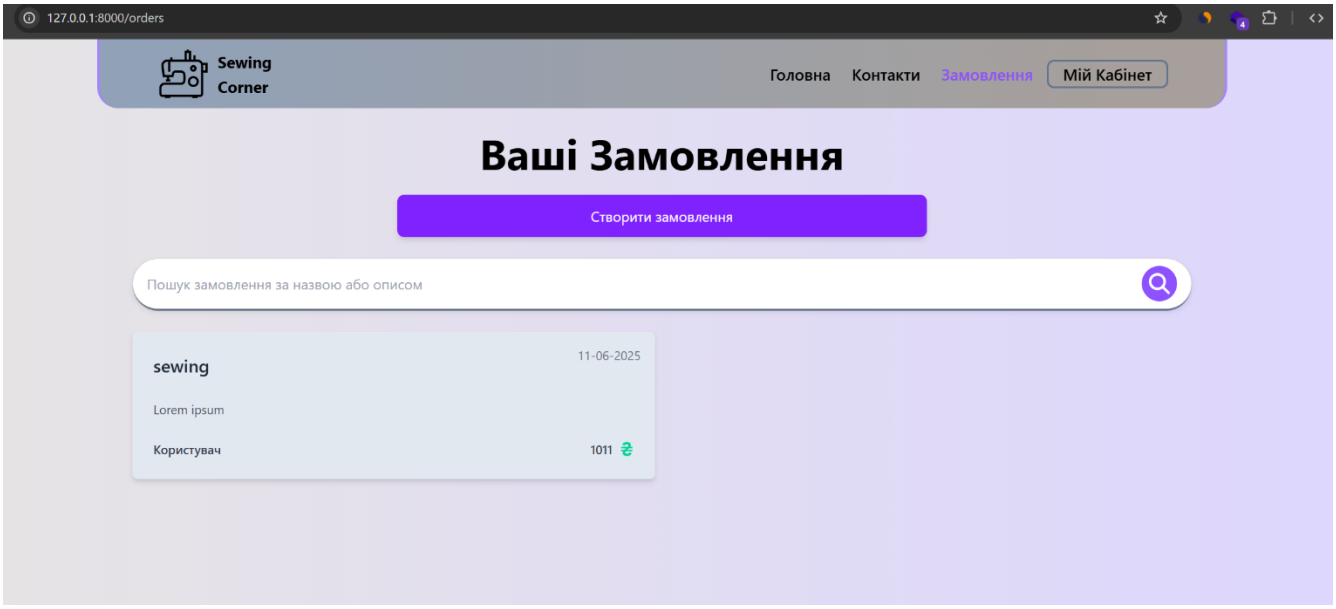

Рисунок 4.2.13 Сторінка з замовленнями

Пошук Замовлення

Компонент order-search.blade.php, розташований за шляхом resources\views\components\order-search.blade.php, реалізує функціональну та естетично оформлену форму пошуку замовлень. Його головне призначення — надати користувачеві змогу здійснювати текстовий пошук за назвою або описом замовлення безпосередньо зі сторінки перегляду замовлень (orders.index).

Форма використовує метод GET, що дозволяє передавати пошуковий запит у вигляді параметра URL, зберігаючи його між запитами. Атрибут action спрямовує запит на маршрут orders.index, до якого прикріплюється параметр search із вмістом, уведеним користувачем. Таким чином, обробка запиту пошуку виконується в тому самому контролері, який відповідає за виведення списку замовлень, і не потребує окремої логіки маршрутизації.

Поле введення — це HTML-елемент <input type="search">, що візуально інтегрується в дизайн без меж (border-none) і зберігає значення пошуку навіть після оновлення сторінки за допомогою виразу value="{{ request('search') }}". Це забезпечує зручність користувача та підтримку контексту пошуку під час навігації:

```
<form action="{{ route('orders.index') }}" class="w-full mx-auto border-b-2  border-slate-500 rounded-full hover:border-violet-700" method="GET">

   <div class="flex items-center gap-2    bg-white border border-stone-300 rounded-full shadow-md px-4 py-2">

      <!-- Поле пошуку -->

      <div class="flex-grow">

         <input

            type="search"

            name="search"

            placeholder="Пошук замовлення за назвою або описом"

            value="{{ request('search') }}"

            class="w-full bg-transparent border-none focus:outline-none focus:ring-0 text-gray-700 placeholder-gray-400"

         >

      </div>


      <button

         type="submit"

         class="flex items-center justify-center rounded-full bg-violet-500 p-2 hover:bg-violet-700 transition duration-300 ease-in-out cursor-pointer"

      >
```

```
            <x-icon name="magnifying-glass" class="size-6 fill-white"/>
        </button>
    </div>
</form>
```

Представлення Замовлення у вигдяі картки

Компонент order.blade.php, розміщений за шляхом resources\views\components\order.blade.php, відповідає за відображення одного замовлення у вигляді картки (Рис. 4.2.14). Його основне призначення — надати стислу інформацію про замовлення та забезпечити швидкий доступ до повної інформації через перехід на відповідну сторінку. Кожна така картка викликається у списку через <x-order :$order />.

Уся картка обгорнута в тег <a>, що веде до маршруту orders.show з передачею ідентифікатора замовлення. Таким чином, вся площа картки є клікабельною, що полегшує навігацію.

Вміст структурується всередині тега <article> і містить заголовок із типом послуги та датою створення, основний текстовий опис, а також нижній блок із інформацією про користувача й ціну. Опис замовлення стисло подається в обмеженому по висоті контейнері, який дозволяє прокручування при потребі.

У нижній частині картки відображається ім’я користувача з іконкою, що дозволяє ідентифікувати автора, та блок ціни. Якщо ціна ще не визначена, система адаптивно показує повідомлення відповідно до ролі користувача: адміністратор бачить нагадування про необхідність її встановлення, а звичайний користувач — повідомлення про очікування оцінки. У випадку наявної ціни вона виводиться поруч із відповідною іконкою.

Компонент використовує стандартизовану стилізацію, спираючись на Tailwind CSS, а для графічних елементів — іконки Blade-компонентів. Він побудований з урахуванням повторного використання, масштабованості та легкого розширення при потребі:

```
<a href="{{ route('orders.show', $order->id) }}" class="block rounded-md bg-slate-200 shadow-md hover:shadow-lg transition-shadow duration-300">
  <article class="flex flex-col p-6 h-full relative">
    <header class="mb-4">
      <h3 class="font-semibold text-xl text-gray-800">{{ $order->service_type }}</h3>
      <span class="absolute top-4 right-4 text- sm text-gray-500">{{ $order->created_at->format('d-m-Y') }}</span>
    </header>
    <p class="font-normal mb-6 mt-2 max-h-[150px] overflow-y-auto text- sm text-gray-600 leading-relaxed">
      {{ $order->description }}
    </p>
    <footer class="flex justify-between items-center mt-auto">
      <div class="flex items-center gap-2">
        <x-icon name="user" class="size-6 text-gray-500"/>
        <span class="text- sm font-semibold text-gray-700">{{ $order->user->name }}</span>
      </div>
      <div class="flex items-center gap-2">
```

```
            @if ($order->price == 0)
                @if (auth()->user()->role === \App\Enums\UserRole::Admin)
                    <span class="text-sm font-semibold text-red-600">Задайте ціну</span>
                    <x-icon name="value" class="size-4 fill-red-600"/>
                @else
                    <span class="text-sm font-semibold text-yellow-500">Очікуйте оцінювання</span>
                    <x-icon name="value" class="size-4 fill-yellow-500"/>
                @endif
            @else
                <span class="text-sm font-semibold text-gray-700">{{ $order->price }}</span>
                <x-icon name="value" class="size-4 fill-emerald-400"/>
            @endif


        </div>
    </footer>
</article>
</a>
```

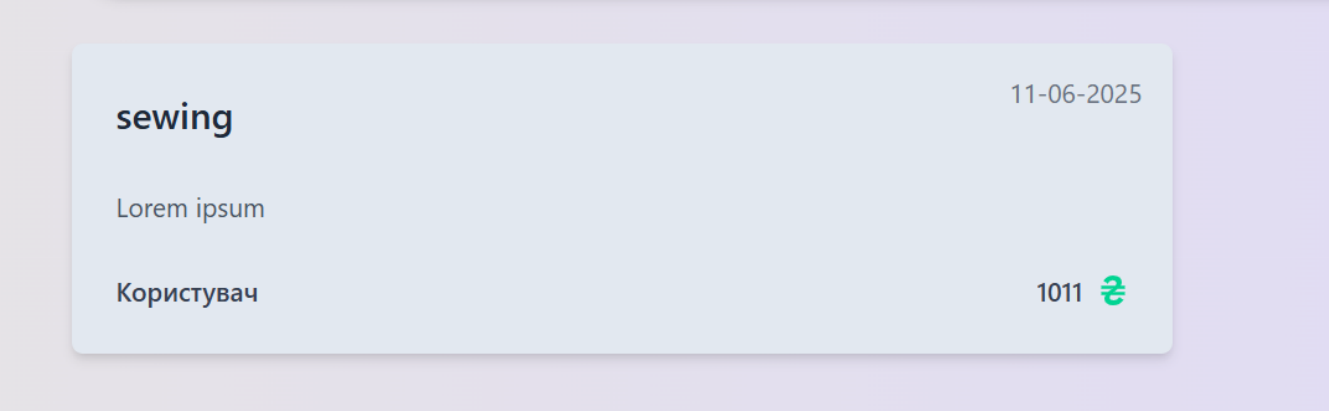

Рисунок 4.2.14 -Замовлення у вигляді картки

Перегляд засмовлення

У файлі resources/views/pages/orders/show.blade.php реалізовано механізм динамічного представлення й обробки даних моделі замовлення (Рис.4.2.15) без звернення до перезавантаження сторінки. Компонент Blade здійснює прив’язку властивостей $order до відповідних місць у розмітці: назва замовлення підставляється в заголовок сторінки, дата останнього оновлення форматується через метод format('d.m.Y'), ім’я користувача–замовника витягується через зв’язок user, статус виконання передається у спеціалізований компонент <x-order-status>, а поля price та payment_status обробляються умовними конструкціями. У разі відсутності значення price виводиться замісний текст «Не вказана», тоді як значення payment_status мапуються на відповідні CSS-класи через Enum-клас \App\Enums\PaymentStatus для відображення семантично коректних кольорових бейджів.

Мультимедійна складова розв’язується за допомогою декодування JSON-масиву, збереженого в полі images. Функція json_decode перетворює рядок у масив шляхів до файлів, після чого кожен шлях у циклі Blade відображається як HTML-елемент <img> з атрибутом onclick="showImageModal(url)". Цей виклик ініціює роботу функції showImageModal(imageUrl), яка програмно змінює класи контейнера модального вікна (#imageModal) із прихованих (hidden) на видимі (flex), а також присвоює атрибут src елементу <img id="modalImage">. Закриття модалки забезпечується функцією

hideImageModal(), що повертає початкові класи й очищує джерело зображення; додатково для зручності передбачено обробник події кліку по напівпрозорому фону, який лише за умови прямого попадання цілі (e.target === this) викликає hideImageModal().

Логіка управління діями над замовленням ґрунтується на зміні CSS-властивостей display та динамічному перемиканні класів. Посилання на маршрут orders.edit передає ідентифікатор замовлення для переходу до сторінки редагування. Виклик openCancelModal(event) блокує стандартну поведінку гіперпосилання та встановлює display: flex для контейнера #cancelModal, де розміщено форму підтвердження скасування з методом POST на маршрут orders.cancel із прихованим полем status="canceled". Функція closeCancelModal() відновлює display: none після відміни дії.

Для ролі адміністратора передбачено додатковий механізм встановлення ціни через окреме модальне вікно #setPriceModal. Виклик openSetPriceModal(event) скасовує стандартну поведінку, перемикає display на flex, усуває класи обмеження прозорості (opacity-0) і взаємодії (pointer-events-none), а також плавно масштабує вміст модалки із scale-90 до scale-100, після чого відбувається програмний фокус на полі введення ціни. Зворотні дії виконує closeSetPriceModal(), яка додає класи непрозорості й блокування взаємодії, повертає масштаб і тільки після затримки приховує вікно. Щоб унеможливити випадкове закриття модального вікна встановлення ціни по кліку на фон, для контейнера встановлено порожній обробник події кліку. Таким чином, поєднання Blade-шаблонів і клієнтських JavaScript-функцій забезпечує гнучке та інтуїтивно зрозуміле оновлення інтерфейсу й взаємодію користувача з деталями замовлення в реальному часі:

```
<x-layout>
    <section class="pt-24 px-6">
        <div class="max-w-6xl mx-auto flex flex-col gap-10 pb-24">
```

```
<h1 class="text-center font-bold text-4xl md:text-5xl text-gray-800 dark:text-gray-100">{{ $order->title }}</h1>

<div class="  bg-gradient-to-br from-slate-300 to-stone-300 rounded-xl shadow-lg p-6 border-b border-slate-200">


    {{-- Інформація про замовлення --}}
    <div class="flex flex-wrap justify-center items-center gap-2 mb-6 text-gray-800">
        <span class="text-xl"><strong>      Створено:</strong> {{ $order->updated_at->format('d.m.Y') }}</span>
        <span class="mx-2 text-gray-400">|</span>
        <span class="text-xl"><strong>      Замовник:</strong> {{ $order->user->name }}</span>
        <span class="mx-2 text-gray-400">|</span>
        <x-order-status :status="$order->status" />
        <span class="mx-2 text-gray-400">|</span>
        <span class="text-xl"><strong>Ціна:</strong>
            @if ($order->price)
                {{ $order->price }} грн
            @else
                <span class="text-red-600">Не вказана</span>
            @endif
        </span>
```

```
                <span class="mx-2 text-gray-400">|</span>

                {{-- Додаємо статус оплати тут --}}

                @if($order->payment_status === \App\Enums\PaymentStatus::Paid)

                    <span class="inline-block px-4 py-2 rounded-full bg-green-100 text-green-800
text-base font-semibold shadow">

                        Оплачено

                    </span>

                @elseif($order->payment_status === \App\Enums\PaymentStatus::Pending)

                    <span class="inline-block px-4 py-2 rounded-full bg-yellow-100 text-yellow-
800 text-base font-semibold shadow">

                        Очікує оплати

                    </span>

                @elseif($order->payment_status === \App\Enums\PaymentStatus::Failed)

                    <span class="inline-block px-4 py-2 rounded-full bg-red-100 text-red-800 text-
base font-semibold shadow">

                        Не вдалося оплатити

                    </span>

                @elseif($order->payment_status === \App\Enums\PaymentStatus::Cash)

                    <span class="inline-block px-4 py-2 rounded-full bg-indigo-100 text-indigo-
800 text-base font-semibold shadow">

                        Готівкою при отриманні

                    </span>
```

```
    @endif
</div>
<p class="text-lg text-center text-gray-700 mb-8 leading-relaxed">
    {{ $order->description }}
</p>
{{-- Галерея зображень --}}
@if ($order->images && is_array(json_decode($order->images)))
    <div class="flex justify-center gap-6 mb-8 cursor-pointer">
        @foreach (json_decode($order->images) as $img)
            <div class="overflow-hidden rounded-xl shadow-md hover:shadow-xl transform hover:scale-[1.03] transition-all duration-300 ease-in-out">
                <img
                    src="{{ asset('storage/' . ltrim($img, '/')) }}"
                    class="w-full h-64 object-cover rounded-xl"
                    alt="Зображення замовлення"
                    onerror="this.src='{{ asset('images/placeholder.png') }}'"
                    style="cursor: zoom-in;"
                    onclick="showImageModal('{{ asset('storage/' . ltrim($img, '/')) }}')"
                >
            </div>
        @endforeach
```

```
            </div>
        @endif
        {{-- Кнопки дій --}}
        <div class="flex flex-wrap justify-center gap-4 mt-6 ">
            <a   href="{{ route('orders.edit', $order->id) }}" class="px-6 py-2 text-black bg-yellow-500 hover:bg-yellow-600 rounded-lg font-semibold shadow transition-all focus:ring-offset-2 transform hover:scale-105 duration-200 cursor-pointer">Змінити</a>

            @if (auth()->user()->role === \App\Enums\UserRole::Admin)
                <button
                    onclick="openSetPriceModal(event)"
                    class="px-6 py-2 text-black bg-blue-600 hover:bg-blue-700 rounded-lg font-semibold shadow transition-all focus:ring-offset-2 transform hover:scale-105 duration-200 cursor-pointer">
                    Задати ціну
                </button>
            @endif
            {{--  Показувати кнопку скасування тільки якщо замовлення не виконано --}}
            @if($order->status !== 'completed' && $order->status !== 'canceled' && $order->status !== 'processing')
                {{-- Кнопка скасування --}}
```

Скасувати

@endif

</section>

{{-- Модальне вікно для збільшення зображення --}}

< img id="modalImage" class="max-w-full max-h-[80vh] object-contain rounded-lg shadow-lg" />

<buttononclick=" hideImageModal()" class="absolute top-2 right-2 text-white text-3xl font-bold p-2 hover:bg-gray-700 rounded-full cursor-pointer transition duration-300 ease-in-out" aria-label="Закрити">×</button>

{{-- Модальне вікно для скасування --}}

```
        <div class="bg-white rounded-xl shadow-lg p-8 max-w-sm w-full text-center">
            <h2 class="text-xl font-bold mb-4 text-gray-800">Скасувати замовлення?</h2>
            <p class="mb-6 text-gray-600">Ви впевнені, що хочете скасувати це замовлення?</p>
            <form method="POST" action="{{ route('orders.cancel', $order->id) }}" class="flex flex-col sm:flex-row justify-center gap-4">
                @csrf
                <input type="hidden" name="status" value="canceled">
                <button type="button" onclick="closeCancelModal()" class="bg-gray-300 hover:bg-gray-400 text-gray-800 px-6 py-2 rounded-lg font-semibold cursor-pointer">
                    Відмінити
                </button>
                <button type="submit" class="bg-red-600 hover:bg-red-700 text-white px-6 py-2 rounded-lg font-semibold cursor-pointer">
                    Так, скасувати
                </button>
            </form>
        </div>
    </div>

    {{-- Модальне вікно для встановлення ціни --}}
```

<divclass=" bg-white rounded-xl shadow-lg p-6 max-w-sm w-full scale-90 transition-all duration-200">

<h2 class="text-lg font-semibold text-gray-800">Встановити ціну</h2>

<button type="button" onclick=" closeSetPriceModal()" class="text-gray-400 hover:text-gray-700 focus:outline-none cursor-pointer">

<svg class="w-6 h-6" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<pathstroke-linecap="round" stroke-linejoin="round" stroke-width="2" d="M6 18L18 6M6 6l12 12"></path>

</svg>

</button>

<form action="{{ route('orders.setPrice', $order->id) }}" method="POST">

@csrf

<label for="price-modal" class="block text- sm font-medium text-gray-700 mb-2">Ціна (грн):</label>

<input

type="number"

id="price-modal"

```
            name="price"
            step="1"
            min="0"
            value="{{ $order->price }}"
            class="block w-full border border-gray-300 rounded-lg px-4 py-2 mb-4 focus:ring-violet-500 focus:border-violet-500 transition-colors duration-200 text-base bg-white text-gray-900"
            required
            placeholder="0"
        >
        @error('price')
            <p class="text-red-500 text-sm mb-2">{{ $message }}</p>
        @enderror
        <div class="flex justify-end gap-3 mt-2">
            <button
                type="button"
                onclick="closeSetPriceModal()"
                class="bg-gray-200 hover:bg-gray-300 text-gray-800 px-5 py-2 rounded-lg font-medium transition-colors duration-200 cursor-pointer">
                Скасувати
            </button>
            <button
```

```
                type="submit"
                class="bg-violet-600 hover:bg-violet-700 text-white px-5 py-2 rounded-lg font-medium shadow transition-all duration-200 cursor-pointer">
                Зберегти
            </button>
        </div>
    </form>
</div>
</div>

<script>
    // Показати модальне вікно зображення
    function showImageModal(imageUrl) {
        const modal = document.getElementById('imageModal');
        const modalImage = document.getElementById('modalImage');
        modalImage.src = imageUrl;
        modal.classList.remove('hidden');
        modal.classList.add('flex');
    }
    // Сховати модальне вікно зображення
    function hideImageModal() {
```

```
    const modal = document.getElementById('imageModal');
    modal.classList.add('hidden');
    modal.classList.remove('flex');
    document.getElementById('modalImage').src = '';
}
// Для скасування
function openCancelModal(event) {
    event.preventDefault();
    document.getElementById('cancelModal').style.display = 'flex';
}
function closeCancelModal() {
    document.getElementById('cancelModal').style.display = 'none';
}
// Додатково: закриття модального вікна по кліку на фон
document.addEventListener('DOMContentLoaded', function() {
    document.getElementById('imageModal').addEventListener('click', function(e) {
        if (e.target === this) hideImageModal();
    });
});
// Функції для модального вікна встановлення ціни
function openSetPriceModal(event) {
```

```
    event.preventDefault();
    const modal = document.getElementById('setPriceModal');
    const modalContent = modal.querySelector('div:first-child');
    modal.style.display = 'flex';
    setTimeout(() => {
        modal.classList.remove('opacity-0', 'pointer-events-none');
        modalContent.classList.remove('scale-90');
        modalContent.classList.add('scale-100');
        document.getElementById('price-modal').focus();
    }, 10);
}
function closeSetPriceModal() {
    const modal = document.getElementById('setPriceModal');
    const modalContent = modal.querySelector('div:first-child');
    modal.classList.add('opacity-0', 'pointer-events-none');
    modalContent.classList.remove('scale-100');
    modalContent.classList.add('scale-90');
    setTimeout(() => {
        modal.style.display = 'none';
    }, 300);
}
```

```
        // Видаляємо закриття по кліку на фон
        document.addEventListener('DOMContentLoaded', function() {
            const modal = document.getElementById('setPriceModal');
            modal.addEventListener('click', function(e) {
                // Нічого не робимо, щоб не закривалось по кліку на фон
            });
        });
    </script>
</x-layout>
```

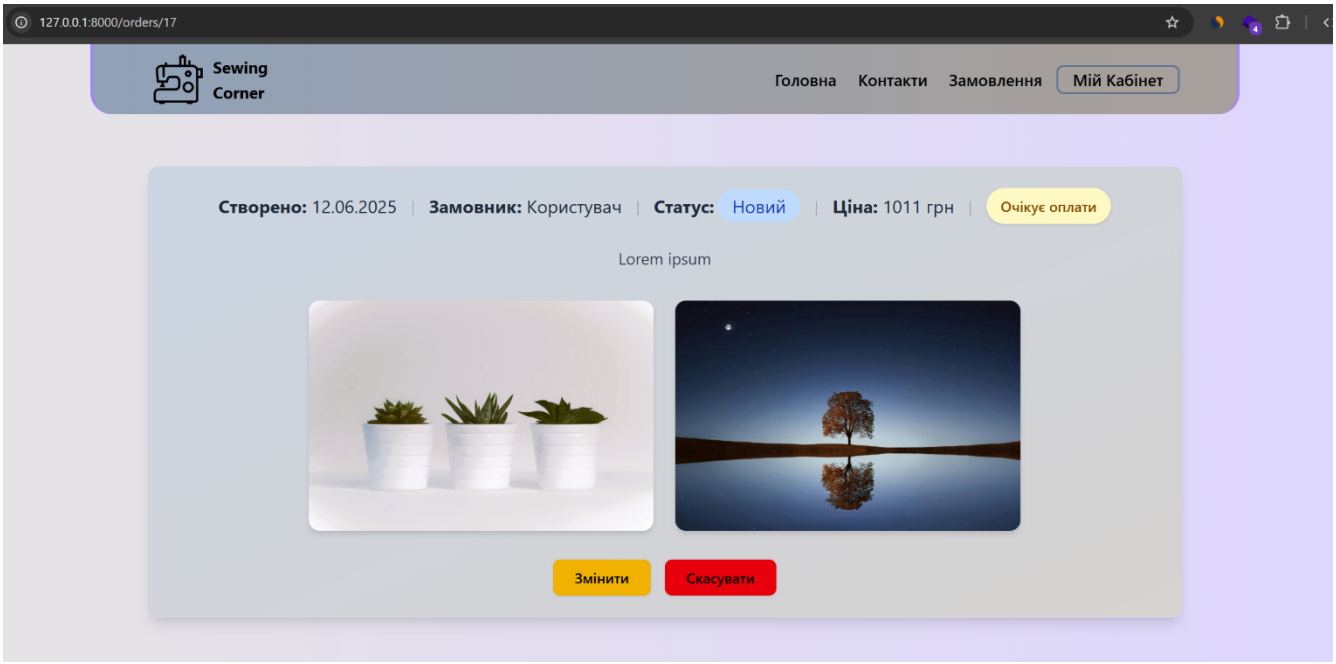

Рисунок 4.2.15 Перегляд замовлення

Кнопка Редагувати Замовлення

Файл resources\views\pages\orders\edit.blade.php реалізує інтерфейс редагування конкретного замовлення (Рис. 4.2.16) шляхом відображення поточних даних моделі

$order і забезпечення можливості їх оновлення безпосередньо в браузері. При завантаженні шаблону перевіряється наявність помилок валідації ($errors->any()), і у разі їх наявності форму обгортання доповнюється блоком, що програмно виводить кожну помилку через цикл @foreach, дозволяючи користувачеві побачити, які саме поля потребують виправлення. Основна форма передбачає метод HTTP PUT на маршрут orders.update, куди потім відправляються всі введені значення.

Для адміністратора доступний випадаючий список зміни статусу замовлення, який формується через Blade-директиву @if(auth()->user()->role === \App\Enums\UserRole::Admin) і містить чотири можливих значення (new, processing, completed, canceled), які за допомогою атрибута @selected набувають стану обраного відповідно до поточного $order->status. Звичайному користувачеві замість цього відправляється приховане поле <input type="hidden" name="status"> із незмінним значенням статусу, що гарантує відсутність несанкціонованих змін.

Селект і textarea для вибору типу послуги (service_type) та редагування опису (description) ініціалізуються поточними значеннями $order->service_type і $order->description, а при відправленні форми ці поля обробляються у контролері із застосуванням валідації і оновлюють відповідні колонки в базі даних.

У правій колонці передбачено механізм оновлення зображень: за допомогою прихованого елемента <input type="file" id="image_upload"> користувач може завантажити до двох нових файлів. Після вибору файлів JavaScript-обробник події change від елемента #image_upload виконує очистку контейнера #new_preview, перевіряє обмеження в два файли і поступово додає до DOM нові прев’ю на основі FileReader.readAsDataURL. Існуючі зображення замовлення декодуються за допомогою json_decode($order->images) і виводяться у контейнері #current_preview до ініціалізації JS-логіки. Таким чином, користувач може одразу побачити зміни у виборі нових зображень перед відправленням форми.

Наприкінці шаблону підключено скрипт, що реалізує логіку контролю кількості файлів, динамічного створення HTML-елементів зображень та показу їх разом з накладами (overlay), у яких відображається номер нового фото. Весь набір операцій на стороні клієнта гарантує інтерактивність процесу редагування без необхідності перезавантажувати сторінку чи додатково підтверджувати кожний крок:

```
<x-layout>
    <div class="flex items-center justify-center min-h-screen p-10 ">
        <div class="max-w-6xl w-full rounded-2xl shadow-xl border border-slate-200 bg-gradient-to-br from-slate-300 to-stone-300  p-8 transition-all duration-300 hover:shadow-2xl">
            <h2 class="text-3xl font-bold mb-6 text-center text-gray-800 border-b pb-4 border-slate-200">Редагування замовлення</h2>

            {{-- Перевірка на наявність помилок --}}
            @if ($errors->any())
                <div class="mb-6 p-4 bg-red-50 border-l-4 border-red-500 rounded-lg">
                    <h4 class="text-red-800 font-semibold mb-2">Помилки при заповненні форми:</h4>
                    <ul class="ml-4 text-red-700 list-disc list-inside">
                        @foreach ($errors->all() as $error)
                            <li>{{ $error }}</li>
                        @endforeach
                    </ul>
                </div>
```

```
@endif

<form method="POST" action="{{ route('orders.update', $order->id) }}" enctype="multipart/form-data">
@csrf
@method('PUT')

<div class="flex flex-col md:flex-row gap-8">
{{-- Ліва колонка: Основна інформація --}}
<div class="w-full md:w-1/2 space-y-5 bg-slate-50 p-6 rounded-xl shadow-inner">
{{-- Поле для статусу тільки для адміністратора --}}
@if(auth()->user()->role === \App\Enums\UserRole::Admin)
<div class="transition-all duration-200 hover:translate-y-[-2px]">
<labelfor="status"class="blocktext- sm font-medium text-gray-700 mb-1">Статус</label>
<select id="status" name="status" class="block w-full border border-gray-300 bg-white rounded-lg shadow-sm focus:ring-violet-500 focus:border-violet-500 px-4 py-3 appearance-none bg-[url('')] bg-no-repeat bg-right-4 bg-center-y">
```

<option value="new" @selected($order->status === 'new') class="py-2">Новий</option>

<option value="processing" @selected($order->status === 'processing') class="py-2">В обробці</option>

<option value="completed" @selected($order->status === 'completed') class="py-2">Виконано</option>

<option value="canceled" @selected($order->status === 'canceled') class="py-2">Скасовано</option>

</select>

@else

{{-- Приховане поле статусу для звичайних користувачів --}}

<input type="hidden" name="status" value="{{ $order->status }}">

@endif

<label for=" service_type" class="block text-sm font-medium text-gray-700 mb-1">Тип послуги</label>

<select id=" service_type" name="service_type" class="block w-full border border-gray-300 bg-white rounded-lg shadow-sm focus:ring-violet-500 focus:border-violet-500 px-4 py-3 appearance-none bg-[url('

nLzIwMDAvc3ZnIj48cGF0aCBkPSJNN7AxMGw1IDUgNS01SDd6IiBmaWxsPSIjNkI3Mjgw Ii8+PC9zdmc+')] bg-no-repeat bg-right-4 bg-center-y">

<option value="repair" @selected($order->service_type === 'repair') class="py-2">Ремонт</option>

<option value="sewing" @selected($order->service_type === 'sewing') class="py-2">Пошив</option>

<option value="other" @selected($order->service_type === 'other') class="py-2">Інше</option>

</select>

<label for="description" class="block text-sm font-medium text-gray-700 mb-1">Опис роботи</label>

<textarea id="description" name="description" rows="8" class="mt-1 block w-full border border-gray-300 bg-white rounded-lg shadow-sm focus:ring-violet-500 focus:border-violet-500 px-4 py-3 resize-none">{{ $order->description }}</textarea>

<button type="submit" class="w-full px-6 py-3 bg-gradient-to-r from-violet-600 to-violet-800 text-white rounded-lg font-semibold shadow-lg hover:from-violet-500

hover:to-violet-700 focus:ring-2 focus:ring-violet-500 focus:ring-offset-2 cursor-pointer transition-all duration-300 transform hover:scale-[1.02] hover:shadow-xl">

Оновити замовлення

</button>

{{-- Права колонка: Зображення --}}

<label class="block text-sm font-medium text-gray-700 mb-1">Оновити зображення</label>

Нові зображення замінять існуючі

<input type="file" id="image_upload" name="images[]" accept="image/*" multiple class="hidden">

<label for="image_upload" class="cursor-pointer px-4 py-3 bg-white hover:bg-gray-50 border border-gray-300 rounded-lg shadow-sm text-center text-slate-700 font-medium transition-all duration-200 transform hover:scale-[1.01] hover:border-violet-400 hover:text-violet-700 hover:shadow-md">

<svg xmlns="http://www.w3.org/2000/svg" class="h-5 w-5" fill="none" viewBox="0 0 24 24" stroke="currentColor">

<pathstroklinecap="round" stroke-linejoin="round" stroke-width="2" d="M4 16v1a3 3 0 003 3h10a3 3 0 003-3v-1m-4-8l-4-4m0 0L8 8m4-4v12" />

</svg>

Обрати файли

</label>

<h4 class="text-sm font-medium text-gray-700 mb-2 flex items-center">

< svg xmlns="http://www.w3.org/2000/svg" class="h-4 w-4 mr-1 text-violet-500" fill="none" viewBox="0 0 24 24" stroke="currentColor">

<pathstroklinecap="round" stroke-linejoin="round" stroke-width="2" d="M4 16l4.586-4.586a2 2 0 012.828 0L16 16m-2-2l1.586-1.586a2 2 0 012.828 0L20 14m-6-6h.01M6 20h12a2 2 0 002-2V6a2 2 0 00-2-2H6a2 2 0 00-2 2v12a2 2 0 002 2z" />

</svg>

Поточні зображення

</h4>

@if ($order->images && is_array(json_decode($order->images)))

@foreach (json_decode($order->images) as $image)

<divclass="relativerounded-lgoverflow-hiddenshadow-mdgroup transition-all duration-300 hover:shadow-lg transform hover:scale-[1.02]">

< img src="{{ asset('storage/' . ltrim($image, '/')) }}" class="w-full h-40 object-cover">

<divclass="absoluteinset-0 bg-gradient-to-t from-black/50 to-transparent opacity-0 group-hover:opacity-100 transition-opacity duration-300 flex items-end">

<spanclass="text-whitetext- xs px-2 py-1 ml-2 mb-2 bg-black/30 rounded">Поточне фото

@endforeach

@else

<divclass="col-span-2flexitems-centerjustify-centerh-40bg-slate-100 rounded-lg text-slate-400 text-sm">

Немає завантажених зображень

@endif

```
                    <h4 class="text-sm font-medium text-gray-700 mb-2 flex items-center">
                        < svg xmlns="http://www.w3.org/2000/svg" class="h-4 w-4 mr-1 text-emerald-500" fill="none" viewBox="0 0 24 24" stroke="currentColor">
                            <pathstroklinecap="round" stroke-linejoin="round" stroke-width="2" d="M12 9v3m0 0v3m0-3h3m-3 0H9m12 0a9 9 0 11-18 0 9 9 0 0118 0z" />
                        </svg>
                        Нові зображення
                    </h4>
                    <div id="new_preview" class="grid grid-cols-2 gap-3 min-h-[100px]">
                        <divclass="col-span-2flexitems-centerjustify-centerh-40bg-slate-50 border border-dashed border-slate-200 rounded-lg text-slate-400 text-sm">
                            Нові фото з'являться тут після вибору
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </form>

    {{-- Блок статусу оплати --}}
    <div class="mt-8 mb-6">
```

@if($order->payment_status === \App\Enums\PaymentStatus::Paid)

< svg class="w-6 h-6 text-green-600" fill="none" stroke="currentColor" viewBox="0 0 24 24">

<path stroke- linecap="round" stroke-linejoin="round" stroke-width="2" d="M5 13l4 4L19 7"></path>

</svg>

Оплачено

@if($order->payment_date)

Дата оплати: {{ $order->payment_date->format('d.m.Y H:i') }}

@endif

@elseif($order->payment_status === \App\Enums\PaymentStatus::Pending)

```
                    <    svg class="w-6 h-6 text-yellow-600" fill="none" stroke="currentColor"
viewBox="0 0 24 24">

                        <pathstroke-    linecap="round" stroke-linejoin="round" stroke-width="2"
d="M12 8v4l3 3m6-3a9 9 0 11-18 0 9 9 0 0118 0z"></path>

                    </svg>

                </div>

                <div>

                    <span class="inline-block px-3 py-1 rounded-full bg-yellow-100 text-
yellow-800 text-sm font-medium">Очікує оплати</span>

                </div>

            @elseif($order->payment_status === \App\Enums\PaymentStatus::Failed)

                <div class="flex items-center justify-center w-12 h-12 rounded-full bg-red-
100">

                    <        svg class="w-6 h-6 text-red-600" fill="none" stroke="currentColor"
viewBox="0 0 24 24">

                        <pathstroke-    linecap="round" stroke-linejoin="round" stroke-width="2"
d="M6 18L18 6M6 6l12 12"></path>

                    </svg>

                </div>

                <div>

                    <span class="inline-block px-3 py-1 rounded-full bg-red-100 text-red-800
text-sm font-medium">Не вдалося оплатити</span>

                </div>
```

```
            @elseif($order->payment_status === \App\Enums\PaymentStatus::Cash)
                <div class="flex items-center justify-center w-12 h-12 rounded-full bg-indigo-100">
                    <    svg class="w-6 h-6 text-indigo-600" fill="none" stroke="currentColor" viewBox="0 0 24 24">
                        <pathstroke-    linecap="round" stroke-linejoin="round" stroke-width="2" d="M17 9V7a2 2 0 00-2-2H5a2 2 0 00-2 2v6a2 2 0 002 2h2m2 4h10a2 2 0 002-2v-6a2 2 0 00-2-2H9a2 2 0 00-2 2v6a2 2 0 002 2z"></path>
                    </svg>
                </div>
                <div>
                    <span class="inline-block px-3 py-1 rounded-full bg-indigo-100 text-indigo-800 text-sm font-medium">Готівкою при отриманні</span>
                </div>
            @endif
        </div>

        @if(auth()->user()->role === \App\Enums\UserRole::Admin)
            <form method="POST" action="{{ route('     orders.updatePaymentType', $order->id) }}" class="mt-4 flex flex-col md:flex-row items-center gap-3">
                @csrf
                @method('PUT')
```

<label for=" payment_type" class="block text-sm font-medium text-gray-700 mb-1 md:mb-0">Статус оплати:</label>

<select id=" payment_type" name="payment_status" class="block w-60 border border-gray-300 bg-white rounded-lg shadow-sm focus:ring-violet-500 focus:border-violet-500 px-4 py-2">

<option value="pending" @selected($order->payment_status === \App\Enums\PaymentStatus::Pending)>Очікує оплати</option>

<option value="cash" @selected($order->payment_status === \App\Enums\PaymentStatus::Cash)>Готівкою при отриманні</option>

<option value="failed" @selected($order->payment_status === \App\Enums\PaymentStatus::Failed)>Не вдалося оплатити</option>

<option value="paid" @selected($order->payment_status === \App\Enums\PaymentStatus::Paid)>Оплачено</option>

</select>

<button type="submit" class="px-5 py-2 bg-blue-600 text-white rounded-lg shadow hover:bg-blue-700 focus:outline-none focus:ring-2 focus:ring-blue-500 transition">

Зберегти зміни

</button>

</form>

@endif

```
<script>
document.getElementById('image_upload').addEventListener('change', function(event) {
    const files = event.target.files;

    if (files.length > 2) {
        alert('Можна завантажити не більше 2 зображень!');
        event.target.value = ''; // Очистити вибір зображень
        return;
    }

    // Очищаємо контейнер для нових зображень
    const newPreviewContainer = document.getElementById('new_preview');
    newPreviewContainer.innerHTML = '';

    // Додаємо нові зображення у превью
    if (files.length === 0) {
        newPreviewContainer.innerHTML = `
            <div class="col-span-2 flex items-center justify-center h-40 bg-slate-50 border border-dashed border-slate-200 rounded-lg text-slate-400 text-sm">
                Нові фото з'являться тут після вибору
```

```
        </div>
    `;
    return;
}

Array.from(files).forEach((file, index) => {
    const imgContainer = document.createElement('div');
    imgContainer.className = 'relative rounded-lg overflow-hidden shadow-md group transition-all duration-300 hover:shadow-lg transform hover:scale-[1.02]';

    const img = document.createElement('img');
    img.classList.add('w-full', 'h-40', 'object-cover');

    const overlay = document.createElement('div');
    overlay.className = 'absolute inset-0 bg-gradient-to-t from-black/50 to-transparent opacity-0 group-hover:opacity-100 transition-opacity duration-300 flex items-end';
    overlay.innerHTML = `<span class="text-white text-xs px-2 py-1 ml-2 mb-2 bg-black/30 rounded">Нове фото ${index + 1}</span>`;

    const reader = new FileReader();
    reader.onload = function(e) {
        img.src = e.target.result;
```

```
        };
        reader.readAsDataURL(file);
        imgContainer.appendChild(img);
        imgContainer.appendChild(overlay);
        newPreviewContainer.appendChild(imgContainer);
    });
});
</script>
</x-layout>
```

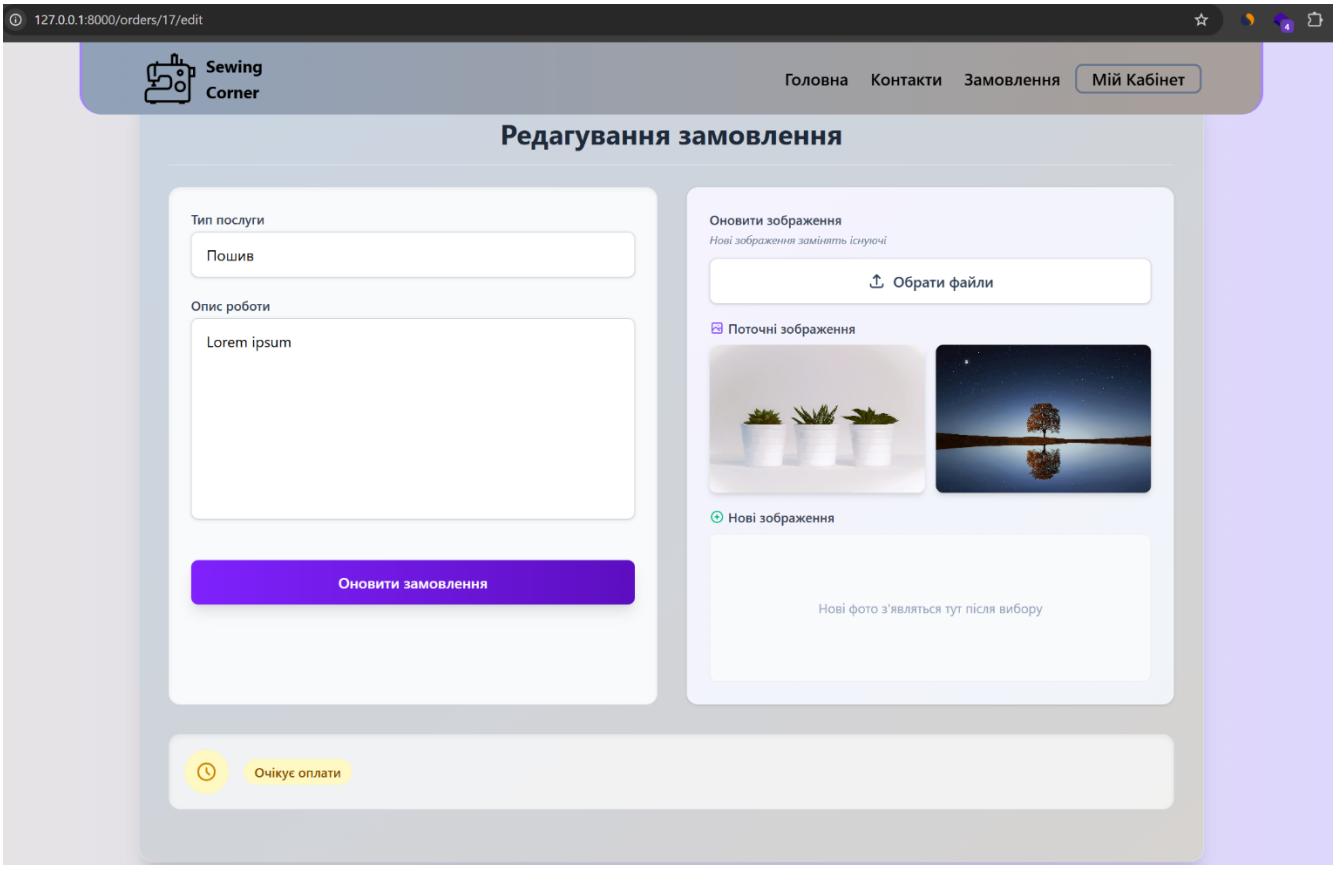

Рисунок 4.2.16 Редагування замовлення

Головна сторінка(авторизований користувач)

Фрагмент шаблону реалізує інтерфейс головної сторінки вебзастосунку для авторизованого користувача (Рис. 4.2.17) в середовищі Laravel з використанням Blade та Tailwind CSS. Компонент розміщено в секції @auth, що гарантує його відображення лише для автентифікованих сесій:

```
@auth
   <section class="flex flex-col gap-8 pt-16 px-10">
      <div class="conteiner flex flex-col gap-8 p-24">
         <h2 class="text-center font-bold text-4xl">Наші останні замовлення</h2>
         <div class="grid grid-cols-2 gap-4">
            @foreach ($orders as $order)
               <x-order :$order/>
            @endforeach
         </div>


         @if (method_exists($orders, 'links'))
            <div class="sm:flex-1 sm:flex sm:items-center sm:justify-between flex-1 flex flex-col items-center bg-white text-red-800 p-4 rounded shadow">
               {{ $orders->links() }}
            </div>
         @endif
      </div>
   </section>
@endauth
```

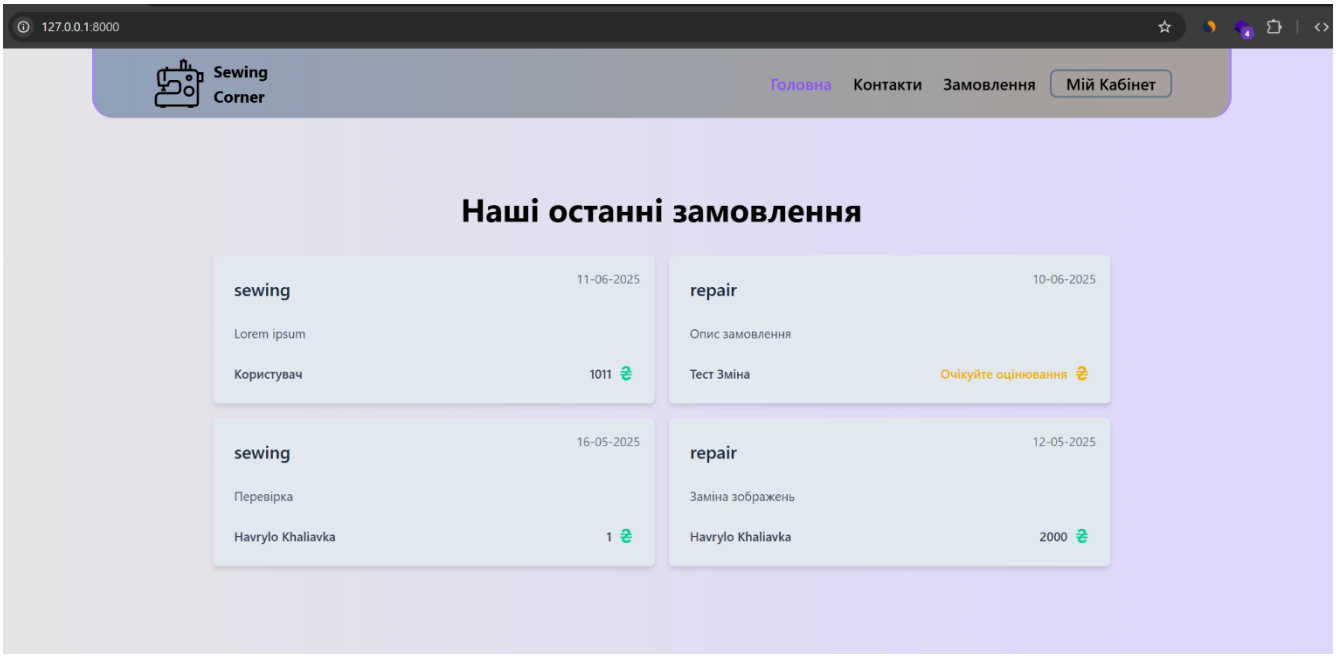

Рисунок 4.2.17 Головна сторінка - вигляд для зареєстрованого користувача

Панель Адміністратора сайту

Панель адміністратора сайту (Рис. 4.2.18) є функціональним інтерфейсом, розробленим для ефективного управління контентом, замовленнями та користувачами вебплатформи. Вона реалізована із застосуванням Blade-шаблонів у фреймворку Laravel і базується на адаптивній сітці з використанням Tailwind CSS, що забезпечує сучасний зовнішній вигляд і зручність у взаємодії.

У центральній частині інтерфейсу розміщується вітальне повідомлення, яке динамічно звертається до користувача на ім’я та підтверджує його адміністративний статус. Ліва колонка призначена для відображення персональної інформації авторизованого адміністратора, включаючи ім’я, електронну пошту, номер телефону та роль у системі, яка маркована відповідним кольором для візуального виділення.

Права частина екрана містить адміністративну панель із трьома основними інтерактивними елементами: переходом до загального списку замовлень, доступом до управління користувачами та створенням нового замовлення. Кожна з кнопок

оформлена у власному кольоровому стилі з ефектами наведеного курсора та анімацією при натисканні, що підвищує інтерактивність і сприйняття користувацького інтерфейсу.

Нижче розміщені окремі дії — редагування профілю та вихід із системи. Вони логічно винесені за межі основної панелі для уникнення змішування адміністративних функцій із персональними. Кожен функціональний блок візуально відокремлений за допомогою тіней, обрамлення та заокруглених кутів, що сприяє кращій читабельності та сприйняттю інформації. Структура побудована відповідно до принципів UX-дизайну, що дозволяє зменшити когнітивне навантаження на користувача та забезпечити інтуїтивну навігацію:

```
@extends('user.dashboard-layout')

@section('subtitle')
   <div class="text-xl text-gray-700 font-medium text-center">
      Тут ви можете керувати всіма замовленнями та користувачами.
      <p class="text-green-600 font-semibold ">Ви адміністратор!</p>
   </div>
@endsection

@section('content')
   <div class="flex flex-col md:flex-row gap-8">
      {{-- Інформація про профіль адміністратора --}}
      <div class="border border-slate-300    bg-white shadow-xl rounded-xl p-6 w-full md:w-1/3">
         <h2 class="text-2xl font-semibold text-gray-800 mb-4 border-b pb-2">          Ваш профіль</h2>
```

```
            <ul class="text-gray-700 space-y-3">
                <li><strong class="text-gray-900">Ім'я:</strong> {{ auth()->user()->name }}</li>
                <li><strong class="text-gray-900">Email:</strong> {{ auth()->user()->email }}</li>
                <li><strongclass="text-gray-900"> Телефон:</strong> {{ auth()->user()->phone ?? 'Не вказано' }}</li>
                <li><strongclass="text-gray-900">Роль:</strong> <span class="text-green-600 font-semibold">Адміністратор</span></li>
            </ul>
        </div>

        {{-- Адмінські блоки  --}}
        <div class="w-full md:w-2/3">
            {{-- Об'єднаний блок управління --}}
            <div class="border border-slate-300 bg-white shadow-xl rounded-xl p-6">
                <h2 class="text-center text-2xl font-semibold text-gray-800 mb-4 border-b pb-2">Адміністративна панель</h2>
                <div class="space-y-4">
                    <a href="{{ route('orders.index') }}"
                    class="w-full block px-6 py-3 bg-blue-600 text-white rounded-lg shadow-md hover:bg-blue-500 focus:ring-2 focus:ring-blue-500 focus:ring-offset-2 text-center transition transform hover:scale-105 duration-300 text-lg">
                        Переглянути всі замовлення
```

```
                </a>

                <a href="{{ route('admin.list') }}"
                class="w-full block px-6 py-3 bg-green-600 text-white rounded-lg shadow-md hover:bg-green-500 focus:ring-2 focus:ring-green-500 focus:ring-offset-2 text-center transition transform hover:scale-105 duration-300 text-lg">
                    Список користувачів
                </a>

                <a  href="{{ route('create-order') }}" class="w-full px-6 py-3 bg-violet-600 text-white rounded-lg shadow-md hover:bg-violet-500 focus:ring-2 focus:ring-violet-500 focus:ring-offset-2 inline-block text-center transition duration-300 ease-in-out transform hover:scale-105">
                    Створити замовлення
                </a>
            </div>
        </div>
    </div>
</div>

{{-- Кнопка редагування профілю --}}
<div class="flex justify-center mt-4">
```

```
        <a href="{{ route('profile.edit') }}"
           class="px-6 py-3 bg-yellow-500 text-white rounded-lg shadow-md hover:bg-yellow-400 focus:ring-2 focus:ring-yellow-400 focus:ring-offset-2 transition transform hover:scale-105 duration-300 text-lg">
            Редагувати профіль
        </a>
    </div>
    {{-- Вихід --}}
    <form action="{{ route('logout') }}" method="POST" class="mt-4 flex justify-center">
        @csrf
        <button
            type="submit"
            class="px-6 py-3 bg-red-600 text-white rounded-lg shadow-md hover:bg-red-500 focus:ring-2 focus:ring-red-500 focus:ring-offset-2 transition transform hover:scale-105 duration-300 text-lg cursor-pointer">
            Вихід
        </button>
    </form>
@endsection
```

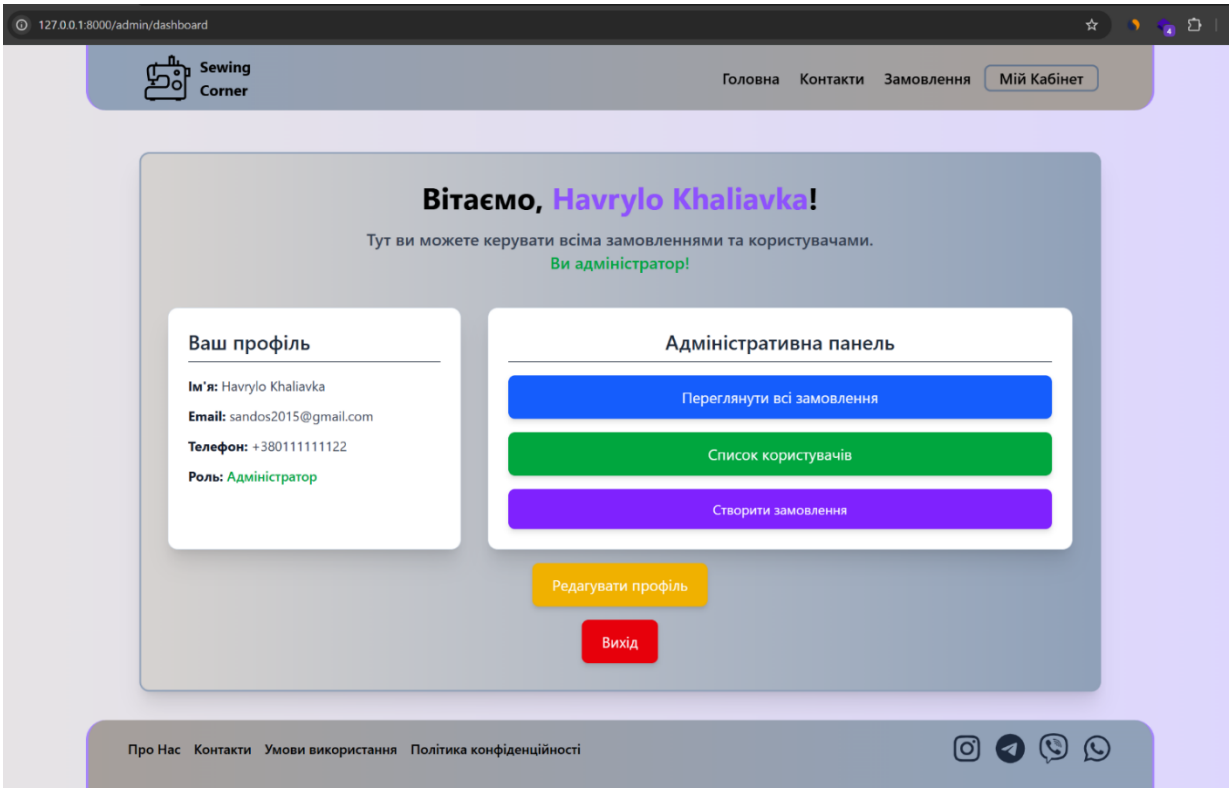

Рисунок 4.2.18 -Адмін Панель

Список Користувачів

Компонент відображення списку користувачів (Рис. 4.2.19) реалізований в файлі resources\views\user\admin\list.blade.php та кастомного компонента <x-layout>, який формує узагальнений макет сторінки. Основна частина вміщена в контейнер з обмеженою максимальною шириною та вираженим градієнтним тлом, що плавно переходить від світло-сірого до бежевого. Візуальне оформлення підсилене заокругленням кутів, глибокими тінями та обрамленням, що створює ефект об’ємності.

У центральній частині представлено табличний інтерфейс, який містить детальну інформацію про зареєстрованих користувачів. Таблиця структурована на основі семи колонок: ID, ім’я, номер телефону, електронна пошта, роль, дата реєстрації та статус верифікації. Всі дані мають чітке візуальне розділення завдяки контрастному

заголовковому рядку темно-сірого кольору з білим текстом, а також чергуванню кольору фону при наведенні на рядок, що покращує читабельність і навігацію..

У верхній частині інтерфейсу чітко відображено з

Особливу увагу приділено відображенню ролі користувача та статусу підтвердження електронної адреси. Значення ролі подано у вигляді тегів з сірим фоном і заокругленням. Статус верифікації електронної пошти реалізований за допомогою кольорових бейджів: зелений фон і напис "Так" для підтверджених користувачів та червоний фон з написом "Ні" для непідтверджених. Це надає швидкий візуальний зворотний зв’язок адміністраторам системи аголовок "Список користувачів", який візуально центровано та стилізовано з використанням великих шрифтів і жирного накреслення. Загальна композиція відповідає сучасним стандартам UI/UX, забезпечуючи одночасно структурованість, контрастність і простоту у взаємодії:

```
<x-layout>

  <div class="container mx-auto max-w-6xl mt-32 px-6 py-10  bg-gradient-to-br from-slate-200 via-slate-300 to-stone-300 rounded-2xl shadow-2xl border border-slate-400">

    {{-- Заголовок --}}

    <h1 class="text-4xl font-extrabold text-center text-slate-800 mb-8 tracking-wide">Список користувачів</h1>


    <div class="overflow-x-auto rounded-lg shadow">

      <tableclass="min-w-fulltext- sm text-left text-gray-800 bg-white rounded-lg overflow-hidden">

        <thead class="text-xs uppercase bg-slate-600 text-white">

          <tr>
```

```
                <th class="px-6 py-4">ID</th>
                <th class="px-6 py-4">Ім'я</th>
                <th class="px-6 py-4">Номер</th>
                <th class="px-6 py-4">Email</th>
                <th class="px-6 py-4">Роль</th>
                <th class="px-6 py-4">Зареєстрований</th>
                <th class="px-6 py-4">Верефікований</th>
            </tr>
        </thead>
        <tbody class="divide-y divide-gray-200">
            @foreach ($users as $user)
                <tr class="hover:bg-violet-100 transition duration-200">
                    <td class="px-6 py-4">{{ $user->id }}</td>
                    <td class="px-6 py-4 font-medium">{{ $user->name }}</td>
                    <td class="px-6 py-4">{{ $user->phone }}</td>
                    <td class="px-6 py-4">{{ $user->email }}</td>
                    <td class="px-6 py-4">
                        <span class="inline-block px-2 py-1 text-xs font-semibold rounded-full
                            {{ $user->role === 'admin' ? 'bg-green-200 text-green-800' : 'bg-gray-
200 text-gray-700' }}">
                            {{ $user->role }}
```

```
                    </span>
                </td>
                <td class="px-6 py-4">{{ $user->created_at->format('d.m.Y H:i') }}</td>
                <td class="px-6 py-4">
                    <span class="inline-block px-2 py-1 text-xs font-semibold rounded-full
                        {{$user-> email_verified_at ? 'bg-emerald-200 text-emerald-800' : 'bg-red-200 text-red-800' }}">
                        {{ $user->email_verified_at ? 'Так' : 'Ні' }}
                    </span>
                </td>
            </tr>
            @endforeach
        </tbody>
    </table>
</div>
</div>
</x-layout>
```

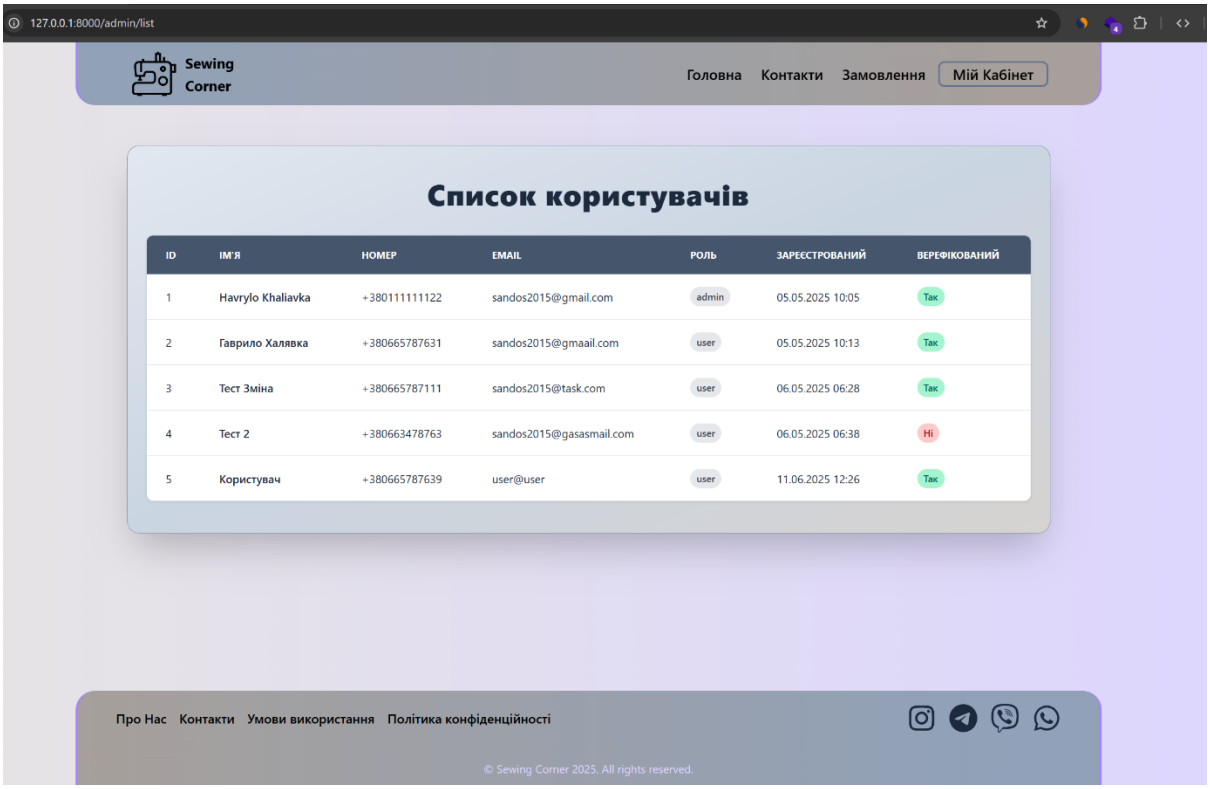

Рисунок 4.2.19 Список користувачів

Керування замовленнями користувачів

Адміністратор у системі має розширені повноваження щодо керування замовленнями користувачів (Рис. 4.2.21). Він отримує доступ до повного переліку замовлень (Рис. 4.2.20), кожне з яких містить детальну інформацію: тип послуги, опис, дату створення, статус виконання, статус оплати, вартість, а також дані про замовника. Адміністратор може здійснювати редагування замовлення, змінювати статус виконання (наприклад, з "Новий" на "Виконано" чи "Скасовано"), а також статус оплати (наприклад, "Очікує оплати", "Оплачено", "Готівкою при отриманні" тощо)(Рис. 4.2.23). Крім того, він має можливість встановлювати або коригувати ціну замовлення (Рис. 4.2.22), скасовувати замовлення, а також переглядати та аналізувати завантажені користувачами зображення, що супроводжують кожне замовлення. Такий функціонал сприяє підтримці рівня контролю та якості обслуговування в системі:

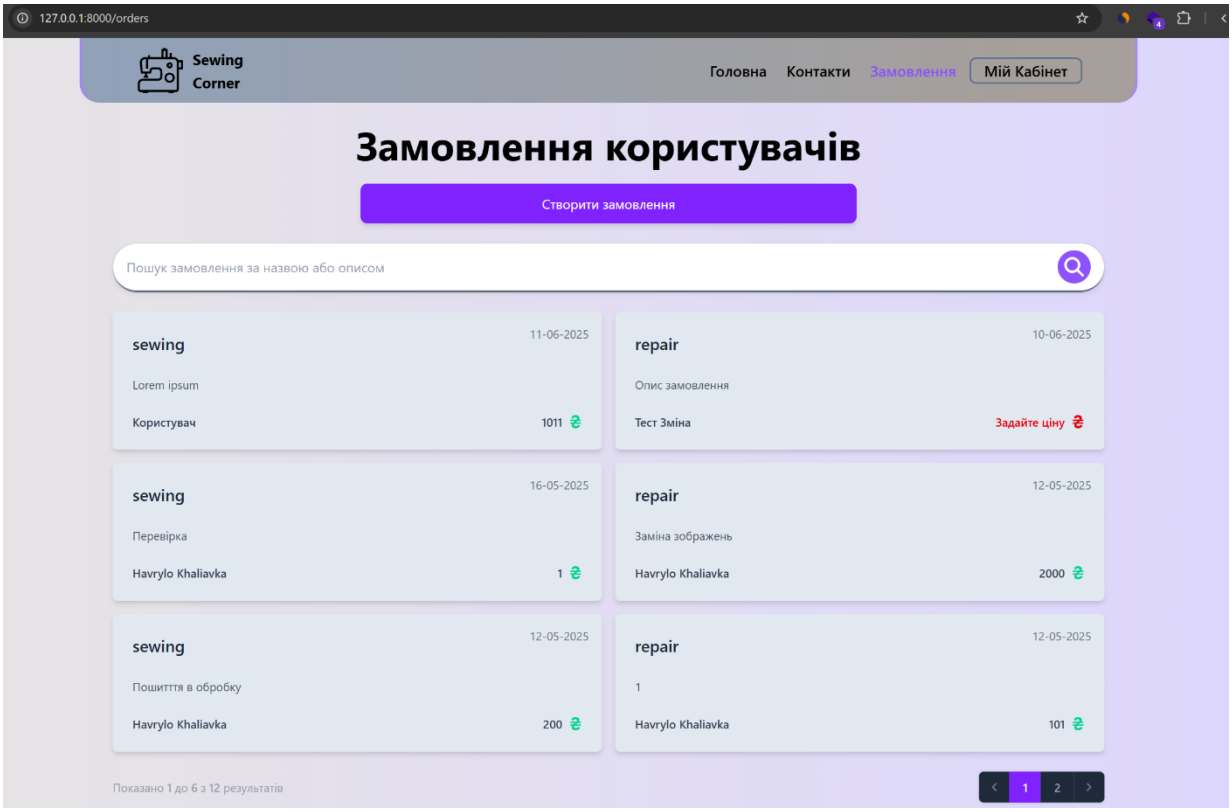

Рисунок 4.2.20 Список зі замовленнями користувачів

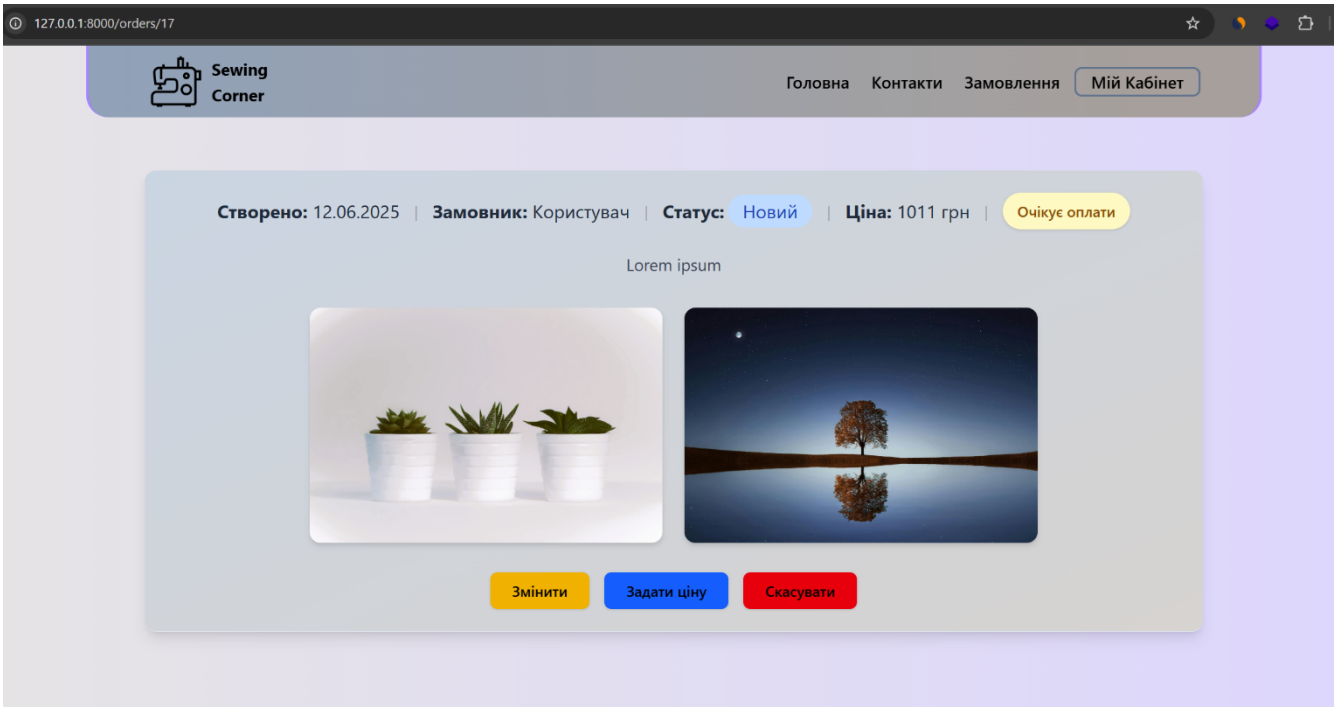

Рисунок 4.2.21 Вигляд замовлення для адміністратора

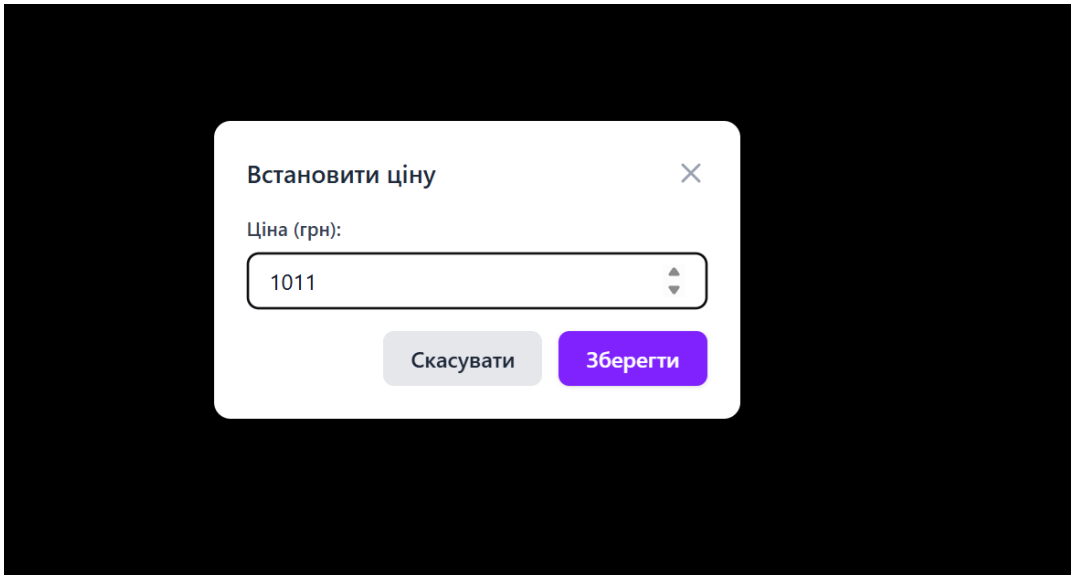

Рисунок 4.2.22 Модульне вікно встановлення ціни

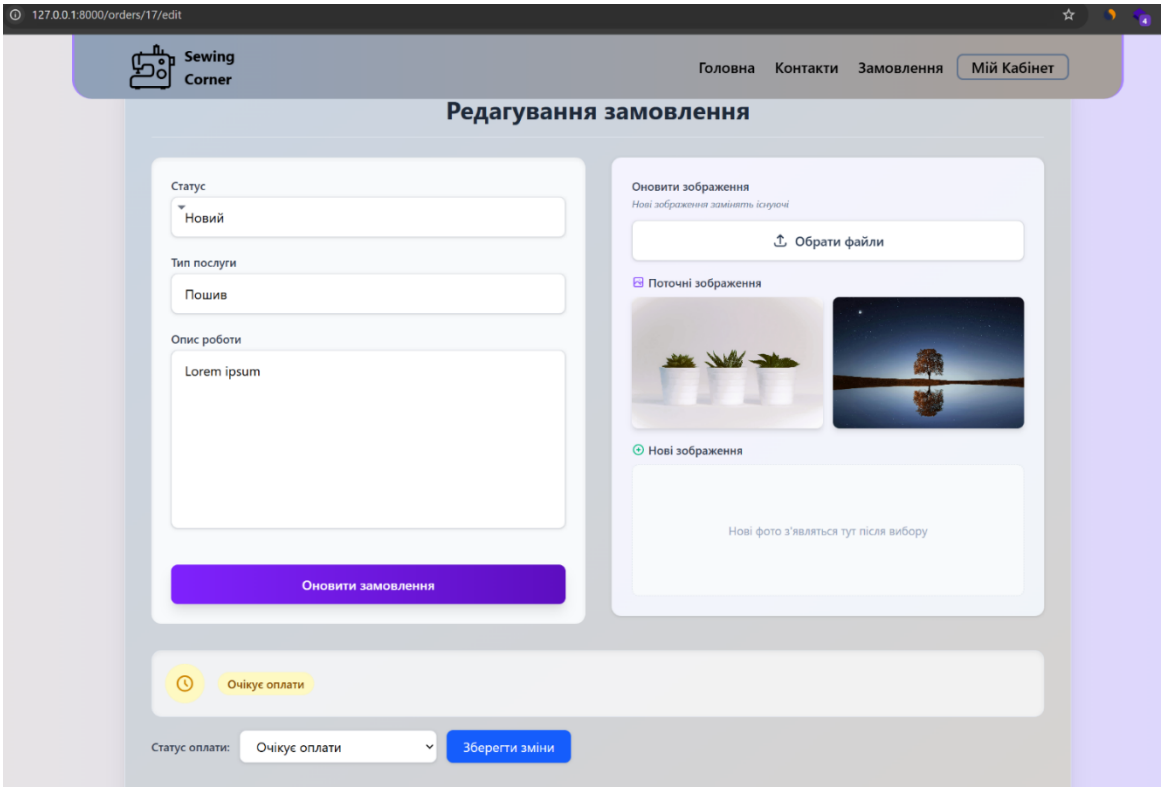

Рисунок 4.2.23 Статус оплати

Відновлення Паролю

Файл forgot-password.blade.php — це Blade-шаблон у проекті, що відповідає за сторінку скидання паролю(Рис 4.2.24). Він містить форму, яка надсилає POST-запит на

маршрут password.email, де обробляється запит на відправку листа з посиланням для скидання. Використано захист CSRF, виведення помилок валідації, поле для введення email з обов’язковим заповненням та стилізацію через Tailwind. Є кнопка "Скинути Пароль" і посилання назад на сторінку входу:

```
<x-layout>
    <div class="flex items-center justify-center min-h-screen">
        <div class="max-w-md w-full p-6 rounded-xl shadow-xl border-2 border-slate-400 bg-gradient-to-l from-slate-400 to-stone-300">
            <h2 class="text-2xl font-bold mb-6 text-center">Скидання Паролю</h2>
            @if ($errors->any())
            <div class="mb-4 text-red-600">
                <ul>
                    @foreach ($errors->all() as $error)
                        <li>{{ $error }}</li>
                    @endforeach
                </ul>
            </div>
            @endif
            <form method="POST" action="{{ route('password.email') }}" autocomplete="off">
                @csrf
                <!-- Email -->
                <div class="mb-4">
```

```
            <labelfor="email"class="blocktext- sm font-medium text-gray-700">Електронна пошта</label>
            <input
                id="email"
                name="email"
                type="email"
                placeholder="Введіть вашу електронну пошту"
                class="mt-1 block w-full border-gray-300 rounded shadow- sm focus:ring-violet-500 focus:border-violet-500 px-4 py-3 bg-white"
                value="{{ old('email') }}"
                autocomplete="off"
                required
            >
        </div>
        <div class="mb-4 flex justify-center">
            <button
                type="submit"
                class="px-4py-2bg-violet-600text-whiteroundedshadowhover:bg-violet-700 focus:ring-2 focus:ring-violet-500 focus:ring-offset-2 cursor-pointer"
            >
                Скинути Пароль
            </button>
```

```
            </div>
            <div class="text-center">
                <a href="{{ route('login') }}" class="text-sm text-violet-500 hover:underline">
                    Повернутися до входу
                </a>
            </div>
        </form>
    </div>
</div>
</x-layout>
```

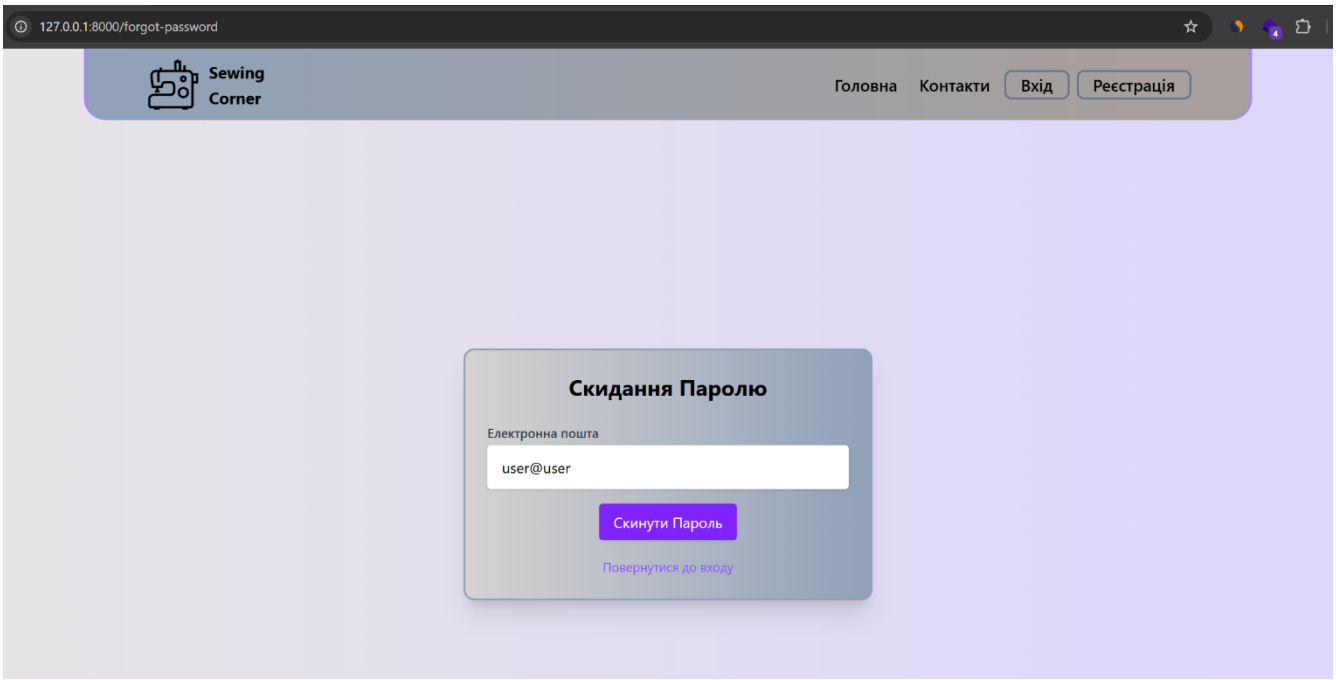

Рисунок 4.2.24 -Форма скидання паролю

Новий пароль

Файл reset-password.blade.php — це шаблон, що відображає форму для встановлення нового паролю(Рис. 4.27). Він приймає токен, отриманий з email(Рис. 4.26), і надсилає POST-запит на маршрут password.update. У формі передбачено введення нового паролю та його підтвердження. Стилізація реалізована через Tailwind CSS. Всі поля обов’язкові до заповнення, а також є механізм виводу валідаційних помилок:

```
<x-layout>
  <div class="flex items-center justify-center min-h-screen">
    <div class="max-w-md w-full p-6 rounded-xl shadow-xl border-2 border-slate-400 bg-gradient-to-l from-slate-400 to-stone-300">
      <h2 class="text-2xl font-bold mb-6 text-center">Скидання Паролю</h2>
      @if ($errors->any())
      <div class="mb-4 text-red-600">
        <ul>
          @foreach ($errors->all() as $error)
            <li>{{ $error }}</li>
          @endforeach
        </ul>
      </div>
      @endif
      <form method="POST" action="{{ route('password.update') }}" autocomplete="off">
        @csrf
```

```
<input type="hidden" name="token" value="{{ $token }}">
 <!-- Новий Пароль -->
<div class="mb-4">
   <labelfor="password"class="blocktext-  sm font-medium text-gray-700">Новий Пароль</label>
   <input
      id="password"
      name="password"
      type="password"
      placeholder="Новий Пароль"
      class="mt-1 block w-full border-gray-300 rounded shadow-  sm  focus:ring-violet-500 focus:border-violet-500 px-4 py-3 bg-white"
      autocomplete="new-password"
      required
   >
</div>
<!-- Повторити Пароль -->
<div class="mb-4">
   <labelfor="password_confirmation" class="block text-sm font-medium text-gray-700">Повторіть Пароль</label>
   <input
```

```
                id="password_confirmation"
                name="password_confirmation"
                type="password"
                placeholder="Повторіть Пароль"
                class="mt-1 block w-full border-gray-300 rounded shadow-sm focus:ring-violet-500 focus:border-violet-500 px-4 py-3 bg-white"
                autocomplete="new-password"
                required
            >
        </div>
        <div class="mb-4 flex justify-center">
            <button
                type="submit"
                class="px-4 py-2 bg-violet-600 text-white rounded shadow hover:bg-violet-700 focus:ring-2 focus:ring-violet-500 focus:ring-offset-2 cursor-pointer"
            >
                Встановити Пароль
            </button>
        </div>
    </form>
</div>
```

```
    </div>

</x-layout>
```

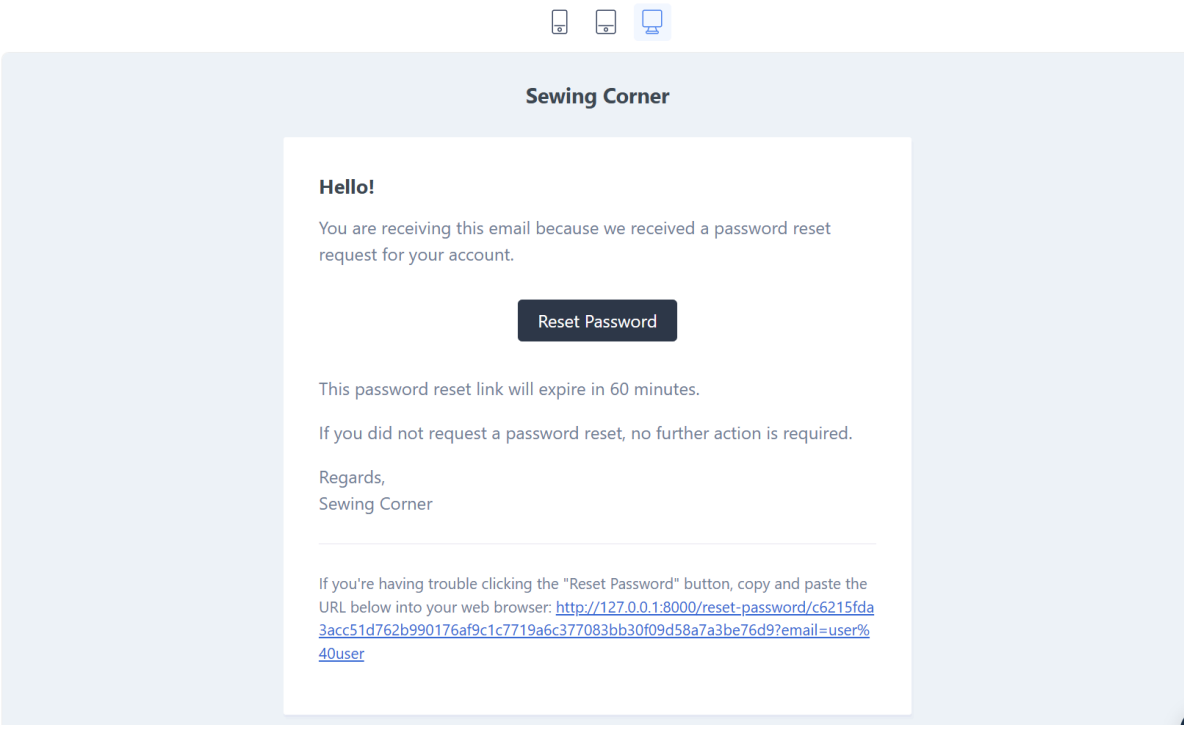

Рисунок 4.26 – Лист відновлення паролю з токеном

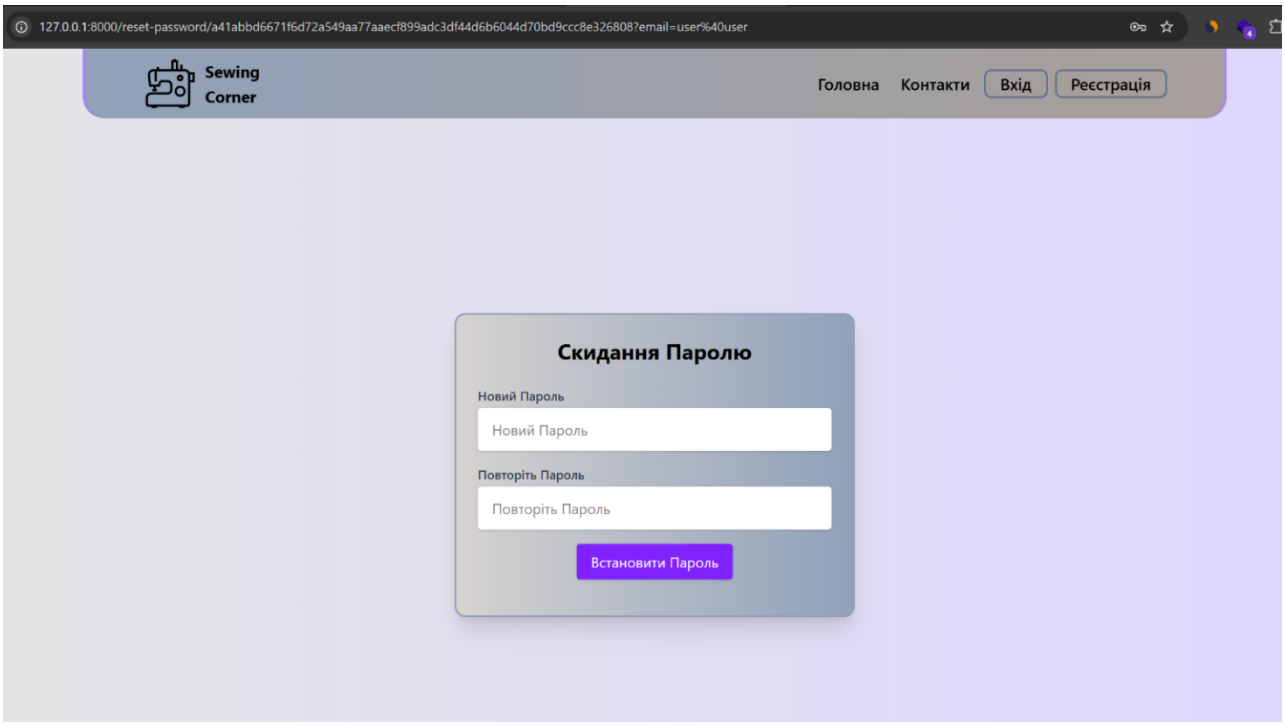

Рисунок 4.27 – Форма встановлення нового паролю

ВИСНОВКИ

У рамках дипломного проєкту було розроблено сучасний вебдодаток для бізнесу з ремонту та пошиття одягу, що базується на фреймворку Laravel 12 та утилітарній CSS-бібліотеці Tailwind. Система була спроектована з урахуванням принципів чистої архітектури: логіка розподілена між контролерами, які відповідають за обробку HTTP-запитів і застосування правил доступу; моделями, що інкапсулюють роботу з базою даних і реалізують предметну логіку (включаючи перерахування станів замовлень та оплат); та поданнями Blade, які забезпечують гнучку адаптивну розмітку інтерфейсу.

При побудові бази даних використано механізм міграцій, що дозволяє відслідковувати еволюцію структури таблиць у системі контролю версій і гарантує узгодженість схеми на різних середовищах. ORM Eloquent надала зручні методи для реалізації реляційних зв’язків та виконання складних вибірок, а застосування PHPenum-типів для статусів спростило перевірку коректності даних та зменшило витрати часу на пошук помилок.

Особливу увагу приділено безпеці: система аутентифікації і верифікації електронної пошти реалізована відповідно до рекомендацій Laravel, механізми скидання пароля та захист від CSRF забезпечують надійність роботи, а кастомні middleware строго розмежовують доступ залежно від ролі користувача. Адміністративний функціонал обмежено лише для користувачів із роллю Admin, при цьому звичайні користувачі мають доступ лише до власного кабінету і власних замовлень.

Інтерфейс додатку концептуально поділений на дашборд користувача та адміністративну панель, кожна з яких містить персоналізовані вітання, блоки інформації та динамічні елементи керування. Tailwind CSS дозволив швидко реалізувати адаптивний дизайн із мінімальним обсягом власного CSS, забезпечивши єдиний стиль і консистентний UX на всіх сторінках.

Робота з медіа—зображеннями виробів та замовлень—організована через зберігання шляхів у JSON-полях моделей, що відкриває можливість подальшого розширення: можна впровадити прев’ю, галереї чи інтеграцію з CDN без зміни базової схеми БД.

Завдяки модульності та масштабованості реалізованої архітектури, додаток може бути легко розширений новими сервісами: онлайноплатою через сторонні API такі як LiqPay, аналітичними панелями, SMS-сповіщеннями або мобільним інтерфейсом. Розроблений підхід до маршрутизації, контролерів і middleware полегшує обслуговування коду та його адаптацію до змінених бізнесвимог, що в сукупності формує сучасну, надійну й безпечну платформу для автоматизації процесів у сфері ремонту та пошиття одягу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Install Tailwind CSS with Laravel Setting up Tailwind CSS in a Laravel project. Режим доступу: URL: https://tailwindcss.com/docs/installation/framework-guides/laravel/vite
2. Ольховська О. Методичні рекомендації щодо оформлення пояснювальних записок до курсових проектів (робіт) для студентів за освітньою програмою «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки та інформаційні технології», «Комп'ютерні науки» галузь знань - 12 «Інформаційні технології». Режим доступу: URL: http://dspace.puet.edu.ua/handle/123456789/12851
3. Laravel DOC Режим доступу: URL: https://laravel.com/docs/12.x
4. Understanding the MVC Architecture in Laravel: A Comprehensive Guide Режим доступу: URL: https://fkrihnif.medium.com/understanding-the-mvc-architecture-in-laravel-a-comprehensive-guide-8f620cc139b6
5. PHP DOC. Режим доступу: URL: https://www.php.net/docs.php
6. Gravelle R. *Best New Features of MySQL 8* (Database Journal) — доклад про можливості MySQL 8. Режим доступу: URL: https://www.databasejournal.com/mysql/best-new-features-of-mysql-8/
7. MySQL DOC. Режим доступу: URL: https://dev.mysql.com/doc/
8. Adekunle M. *Laravel Fundamentals: Understanding the MVC Architecture* (LinkedIn) — огляд ролей Model, View, Controller. Режим доступу: URL: https://www.linkedin.com/pulse/laravel-fundamentals-understanding-mvc-architecture-saheed-ibrahim--mgoyf
9. Understanding Laravel Blade and How To Use It. Режим доступу: URL: https://kinsta.com/blog/laravel-blade/
10. Table Plus DOC. Режим доступу: URL: https://docs.tableplus.com/
11. Icons for your websites. Easy. Done. Режим доступу: URL: https://fontawesome.com/

12. Mailtrap PHP integration. Режим доступу: URL: https://help.mailtrap.io/article/136-mailtrap-php-integration?_gl=1*1t70rm1*_gcl_au*MzY3Nzg4MDk5LjE3NDQ4OTU0Njc.
13. 12 web design best practices for 2024. Режим доступу: URL: https://tillerdigital.com/blog/12-web-design-best-practices-for-2024/