Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Собчука Антона Вікторовича
Прізвище, ім'я, по-батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Вебзастосунок для продажу освітлювальних приладів
Назва теми

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

Шифр КПП.2201125.01.05.ПЗ

Виконав студент III курсу група ІПЗс-22-1 _____ Антон СОБЧУК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник асистент _____ В'ячеслав БОЙКО
Науковий ступінь, звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. тех. наук, доцент _____ Оксана ЯШИНА
Науковий ступінь, звання Підпис Ім'я, ПРІЗВИЩЕ

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

11 червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
2.02.25 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Собчуку Антону Вікторовичу
Прізвище, ім'я, по батькові студента

1. Тема кваліфікаційної роботи Вебзастосунок для продажу освітлювальних приладів
Керівник проєкту (роботи) Бойко В'ячеслав Олександрович, асистент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання
Затверджена наказом ректора університету від 07.02.2025 р. № 23
2. Строк подання студентом роботи на кафедру 01.06.2025 р.
3. Вихідні дані до роботи Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація, тестування програмного забезпечення
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди, 15 шт.)

6. Консультанти розділів дипломного проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Лишка О.Н. доцент кафедри ІПЗ		
Антиплагіат	Форсун Ю.В. доцент кафедри ІПЗ		

7. Дата видачі завдання « 07 » 02 2025р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Ознайомлення з тематикою кваліфікаційних робіт (КвР), визначення та узгодження індивідуальної теми	01.12 – 31.12.2024	
2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2025	
3 Проектування програмного забезпечення	21.02 – 20.03.2025	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2025	
5 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог стандартів	01.05 – 25.05.2025	
6 Попередній захист КвР	Травень 2025	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2025	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2025	

Студент _____ Антон СОБЧУК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ В'ячеслав БОЙКО
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Курсовий проєкт на тему: «Вебзастосунок для продажу освітлювальних приладів»

Автор проєкту: Собчук Антон Вікторович.

Керівник проєкту: Бойко В'ячеслав Олександрович.

Пояснювальна записка: 64 ст., 25 рис., 4 дод., 9 джерел.

Графічна частина: 15 слайдів.

Метою дипломного проєкту є розробка вебдодатку інтернет-магазину з продажу освітлювальних приладів, який дозволяє автоматизувати процеси управління товарами, замовленнями та клієнтами, а також забезпечити можливість подальшої модифікації й масштабування системи.

У роботі застосовано такі інформаційні технології та інструменти: фреймворк Laravel, мови програмування PHP та JavaScript, середовище розробки PhpStorm, система контейнеризації Docker, система управління базами даних MySQL, а також менеджер залежностей Composer.

У процесі виконання проєкту були реалізовані такі етапи:

- проєктування та нормалізація бази даних;
- реалізація архітектурного патерну MVC;
- тестування для виявлення помилок;
- налагодження системи та повторне тестування;
- оформлення пояснювальної записки.

У результаті роботи розроблено вебдодаток, що забезпечує базову функціональність інтернет-магазину, включаючи адміністративну панель для керування товарами, користувачами та замовленнями. Застосування сучасних технологій дозволяє легко масштабувати програму та інтегрувати її з іншими сервісами.

Практична значимість полягає в можливості використання розробленого додатку малими та середніми підприємствами у сфері роздрібної торгівлі.

Система може бути адаптована для компаній з різною спеціалізацією, що робить її універсальним інструментом для управління бізнес-процесами.

У висновку, реалізоване програмне забезпечення є ефективною основою для подальшої модернізації в повнофункціональну платформу з широкими можливостями інтеграції. У майбутньому можливе розширення функціоналу за рахунок впровадження модуля аналітики, підтримки декількох мов та мобільного інтерфейсу.

Ключові слова: ІНТЕРНЕТ-МАГАЗИН, ЛАРАВЕЛ, MYSQL, PHP, СИСТЕМА УПРАВЛІННЯ, ВЕБДОДАТОК

10.06.25
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КПП.2201125.01.05.ПЗ	Пояснювальна записка	89		
2	А4		Завдання на кваліфікаційну роботу	1		
3	А4		Анотація	1		
			Графічні документи			
4	А4		Презентаційні матеріали	15		
5	А4	КПП.2201125.01.05.Е8	Схема бази даних	1		
6	А4	КПП.2201125.01.05.Е8	Схема маршрутизації клієнтської частини	1		
7	А4	КПП.2201125.01.05.Е8	Діаграма класів	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КПП.2201125.01.05.ПЗ		
Розробив		Собчук А.В.		01.06	Вебзастосунок для продажу освітлювальних приладів	Літ.	Арк. 1 / Аркушів 1
Керівник		Бойко В.О.		01.06			
Н.контр.		Яшина О.М.		01.06		ХНУ.ІПЗс-22-1	
Зав. Каф.		Бедратюк Л. П.		01.06			

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

SPA – Single Page Application

БД – База даних

MVCS – Model-View-Controller-Service

REST – Representational State Transfer

ВСТУП

Сьогодні ми живемо у час стрімких змін, коли інформаційні технології та глобалізація економіки повністю змінили наше життя. Електронна комерція стала не просто трендом, а невід'ємною частиною бізнесу і повсякденності. Онлайн-магазини тепер – це потужний інструмент для бізнесу, який дає підприємцям можливість легко знаходити клієнтів по всьому світу, зменшувати витрати та пропонувати покупцям зручний спосіб придбати потрібні речі.

Аналітика провідних компаній підтверджує, що світовий ринок онлайн-торгівлі постійно зростає, показуючи близько 20% зростання щороку протягом останніх п'яти років. Пандемія COVID-19 тільки підштовхнула цей процес, показавши і бізнесу, і покупцям, що цифрові продажі – це вже не просто зручна альтернатива, а часто єдиний вихід. В Україні онлайн-ринок теж не відстає, зростаючи на 15-30% щороку. Це говорить про величезний потенціал для розвитку.

Розвиток онлайн-торгівлі штовхає вперед й інші сфери: логістику (з'являються розумні склади та системи відстеження), фінансові технології (нові безпечні способи оплати) та маркетинг (аналіз даних для персональних пропозицій). Це такий собі ефект доміно, який рухає всю цифрову економіку і задає нові правила гри для бізнесу.

Серед усіх ніш онлайн-ринку, одна з найперспективніших – це товари для дому та інтер'єру. Цей сегмент стабільно росте на 10-15% щорічно. І тут особливу увагу привертають освітлювальні прилади – лампи. Вони виконують не тільки функцію освітлення, а й відіграють ключову роль у дизайні будь-якого приміщення. Світло створює атмосферу, впливає на наш настрій і комфорт. Тому люди тепер шукають не просто «щоб світило», а справжні дизайнерські речі, що пасуватимуть до їхнього стилю та потреб.

Онлайн-магазин для продажу ламп має безліч у порівнянні із звичайними магазинами: можна показати величезний асортимент, який не поміститься у звичайному магазині; показати, як лампи виглядають у різних інтер'єрах; надати всю технічну інформацію; легко порівняти характеристики та ціни; магазин доступний 24/7 з будь-якого куточка світу; і, звісно, це допомагає зробити логістику більш оптимізованою.

Створення онлайн-магазину ламп відповідає сучасним трендам. Люди прагнуть зробити свій дім затишнішим та більш функціональним, особливо після пандемії, коли всі зрозуміли, як багато часу ми проводимо вдома. Ринок домашнього декору продовжує стабільно зростати, підтверджуючи актуальність цього напрямку.

Тому мета розробки онлайн-магазину «Lampach» – це не просто зробити зручний сайт для продажу ламп, а створити повноцінне цифрове рішення, яке допоможе людям легко обрати ідеальне освітлення для свого дому.

В ході виконання дипломного проєкту буде виконано такі завдання:

– проаналізувати ринок ламп та вже існуючих рішень по продажу ламп, визначити їх плюси та недоліки;

– визначити основні функціональні особливості для реалізації дилпомного проекту;

– вибрати та обґрунтувати засоби розробки, поставити вимоги до проекту, що розробляється;

– розробити ПЗ відповідно до вимог, що були поставлені;

– виконати тестування проекту.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Інтернет-магазин (або Ecommerce) [1] – вебсайт, застосунок або цифрова платформа, яка дозволяє бізнесам або окремим особам продавати товари чи послуги через Інтернет. Клієнти можуть переглядати онлайн-каталог, обирати товари, які хочуть придбати, і завершувати транзакції в електронному форматі.

Онлайн-магазини ламп – це швидкий, часто економічно вигідний і доступний спосіб придбання освітлювальних приладів для бізнесу чи особистого використання. Користувачі можуть переглядати та замовляти лампи з будь-якого місця, якщо у них є підключення до Інтернету.

Маркетплейс [2] – це торговельний майданчик, який дає можливість стороннім продавцям реалізовувати товари через свій сайт. Товари завантажуються у загальний каталог маркетплейсу та розміщуються поруч з пропозиціям інших підприємців

Серед основних функціональних елементів інернет-магазинів можна виокремити:

– каталог товарів із категоріями, підкатегоріями, фільтрами та пошуком;

– сторінку товару з фото, описом, характеристиками та наявністю;

– кошик покупця, де зберігаються обрані товари перед оформленням замовлення;

– сторінка з оформленням замовлення з вибором способу доставки та оплати;

– панель адміністратора для керування товарами, замовленнями та користувачами;

– особистий кабінет користувача з історією замовлень;

– система повідомлень та зворотного зв’язку;

– розділ FAQ (часті запитання), де розписані основні моменти щодо доставки товару, оформлення замовлення та інше;

– можливість залишати відгуки та оцінки до товарів користувачами, які вже раніше придбали цей товар.

Наразі авторизація на сайтах Ecommerce відбувається здебільшого на великих маркетплейсах, так як там потрібно зберігати більше інформації про користувача, ніж на сайтах, які представляють лише один тип товарів чи послуг.

На таких сайтах, ідентифікація користувача відбувається через генерацію спеціального токену, який зберігається одночасно на стороні клієнту та на стороні серверу. На рисунку 1.1 зображено принцип роботи таких ідентифікаційних токенів:

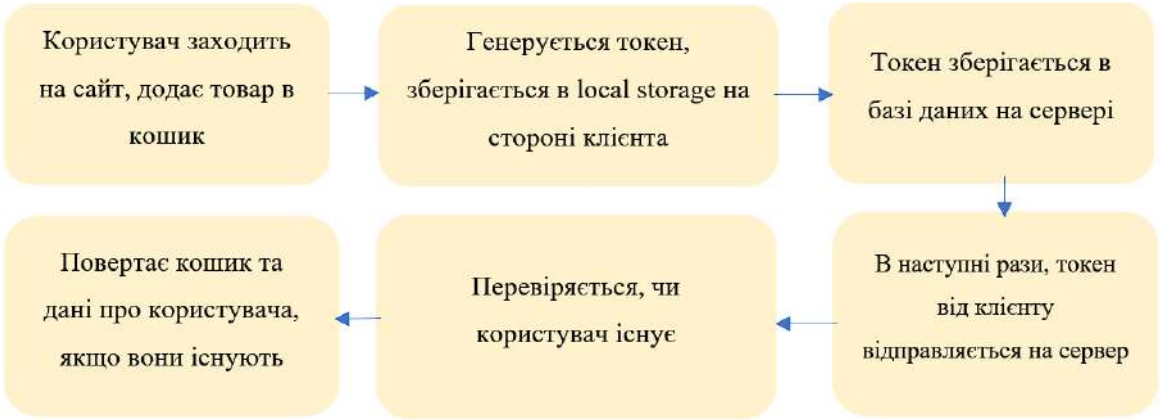


Рисунок 1.1 – Принцип роботи ідентифікаційного токену

Переваги такого типу ідентифікації:

– простота реалізації: логіка токенів легко інтегрується у фронтенд і бекенд, особливо при використанні REST API;

– зручність зберігання: Local Storage зберігає дані навіть після оновлення сторінки чи перезапуску браузера, тому токен не зникає після перезавантаження;

– відокремлення клієнта і сервера: така схема ідеально підходить для SPA-додатків, де фронтенд і бекенд працюють як окремі частини;

– швидкий доступ до токену: доступ до Local Storage – синхронний і миттєвий, не потребує додаткових запитів до сервера.

–

Недоліки такого типу ідентифікації:

– небезпека XSS-атак: Local Storage доступний з JavaScript. Якщо сайт має вразливість до XSS (впровадження шкідливого JS-коду), зловмисник може легко викрасти токен;

– відсутність автоматичної відправки: на відміну від cookie, токени в Local Storage не передаються автоматично з кожним запитом – їх треба додавати вручну в заголовки;

– немає захисту від CSRF-атак, але токен у Local Storage не піддається CSRF-атакам напряму, однак якщо зловмисник отримає токен через XSS, він може виконувати дії від імені користувача.

Так як буде використовуватись такий тип ідентифікації лише для відображення кошуку користувача, це не буде становити загрозу витоку чутливих даних.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Щоб виконати якісний продукт, потрібно дослідити та проаналізувати доступні варіанти в сфері інтернет-магазинів по продажу ламп. Це довзолить визначити найкращий функціонал для дипломного проекту, виявити типові помилки, зрозуміти, чого бажають бачити користувачі на сайтах такого типу. Потрібно провести огляд за такими критеріями:

– огляд вебсайту існуючого рішення: структура інтерфейсу, навігація, UI/UX рішення, реалізація картки товару

– зручність для користувача: адаптивний дизайн під різні пристрої (ПК, телефон, планшет), зрозуміла структура категорій, фільтрація товарів, просте оформлення замовлення

«Розетка» – це один із найбільших українських інтернет-маркетплейсів, який функціонує як багатофункціональна онлайн-платформа для продажу товарів різних категорій, таких як електроніка, побутова техніка, товари для дому, освітлювальні приладів тощо. Сайт поєднує в собі функції віртуального магазину, системи управління замовленнями, платіжного сервісу та логістичної служби.

На рисунку 1.2 зображена головна сторінка сайту «Розетка»

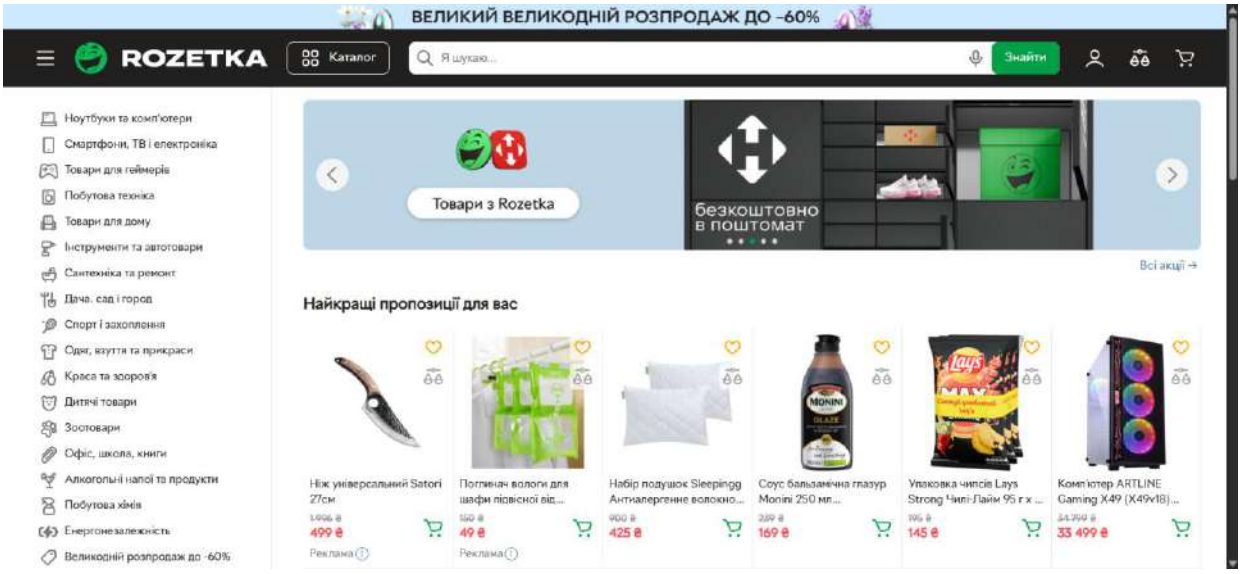


Рисунок 1.2 – Головна сторінка «Розетки»

Одією із головних переваг «Розетки» є широкий асортимент вибору товарів, тобто кінцевий користувач зможе знайти будь-який товар в одному місці і збільшує охоплення сайту в цілому.

Інтуїтивно зрозумілий інтерфейс полегшує навігацію по сайту, пошук потрібних товарів, перегляд інформації та оформлення замовлення. Адаптивність забезпечує комфортне використання сайту на різних пристроях (комп'ютери, планшети, смартфони). Новим покупцям легше зорієнтуватися на сайті, що підвищує ймовірність здійснення покупки.

З ефективною системою фільтрів, що присутня на «Розетці», користувач може швидко знаходити потрібні товари за критеріями (ціна, бренд, характеристики тощо), тим самим зменшуючи свій витрачений час на вибір товару.

Для покупців також корисна функція з багатофункціональною системою відгуків, тобто кожен користувач може ще додатково визначати якість товару за допомогою людей, які вже купили цей товар раніше і вирішили висловити свою позитивну чи негативну думку щодо нього.

На «Розетці» є інтеграція з сервісами по доставці, такими як «Укрпошта», «Нова Пошта», «Meest», а також існують власні точки видачі товарів, що дозволяє відправляти товари у будь-яку точку України.

Із мінусів, «Розетка» не має високою спеціалізації в галузі освітлення. Оскільки вона є універсальним маркетплейсом, категорія освітлення може не отримувати такої ж уваги та спеціалізованої підтримки, як у магазину, що повністю зосереджений на лампах.

Через це випливає і інший недолік, такий як обмежені технічні характеристики вузькоспеціалізованих товарів, у цьому випадку – ламп. Наприклад, індекс передачі кольору (CRI), кут розсіювання світла, тип світлодіодів може бути відсутніми у списку характеристик товару.

Обов'язкова реєстрація може бути мінусом для деяких потенційних покупців, які не хочуть витрачати час на створення облікового запису, особливо якщо вони роблять покупку вперше або є випадковими відвідувачами.

Велика кількість інформації, блоків з рекомендаціями, відгуків та інших елементів на сторінці товару може відволікати увагу покупця від основної інформації про лампу та ускладнювати прийняття рішення про покупку.

На рисунку 1.3 зображена сторінка сайту Intellect-ua.com з каталогом товарів на ПК, а на рисунку 1.4 – на мобільному пристрої. Це рішення є вже вузькоспеціалізованим, де продаються лише лампи для дому чи офісу.

Користувачам, які шукають саме електротехнічну продукцію, буде легше орієнтуватися на сайті, оскільки немає відволікаючих категорій товарів. Сайт є привабливим для ока, з підтримкою мобільного інтерфейсу, що є важливим у наш час, оскільки більшість покупок та відвідувань сайту відбувається з мобільних телефонів (близько 78% всіх покупок відбувлось через мобільні телефони) [3] і з кожним роком лише збільшується.

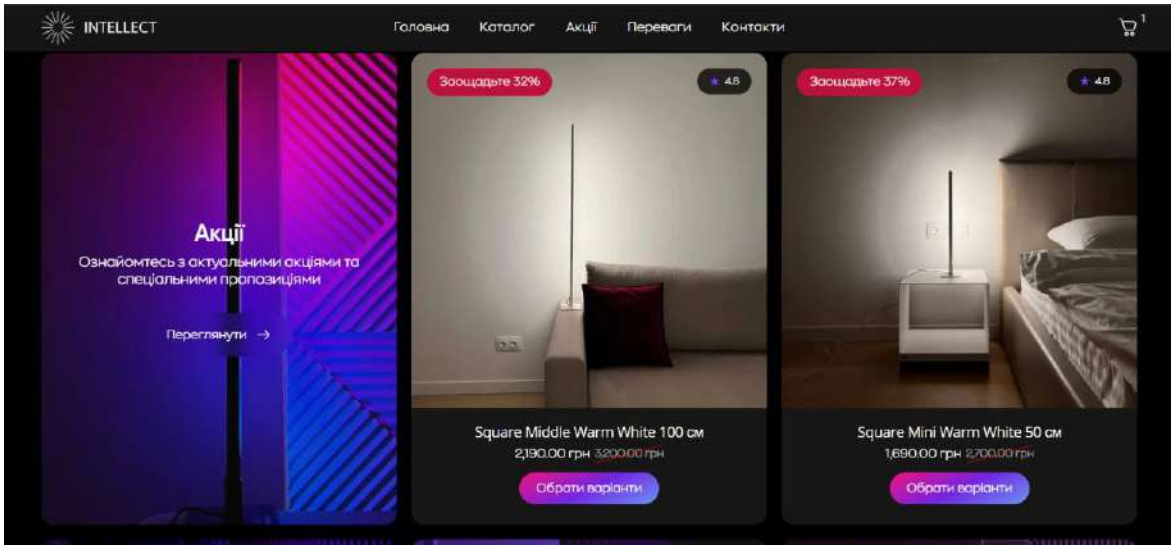


Рисунок 1.3 – Сторінка Intellect-ua.com з каталогом товарів

Рисунок 1.4 – Сторінка з каталогом товарів на мобільному пристрої

Детальні технічні характеристики – як тип цоколя, потужність, температура світла, напруга чи IP-рейтинг – справді важливі для професіоналів, а також для досвідчених користувачів, які розбираються в цих параметрах, тобто вони зможуть зробити більш обґрунтований вибір. Також це дозволити зменшити кількість повернень товарів, оскільки покупці будуть ліпше розуміти, що саме вони купують.

Оскільки «Intellect» є виробником цих ламп і власники отримують прибуток саме від їх продажу – на сайті відсутня реклама, фокус користувача йде лише на товари, які присутні в каталозі та його нічого не відволікає. В теорії, через це підвищується швидкість завантаження сайту, що теж є плюсом.

Оплата товару здійснюється за допомогою «MonoPay» або «LiqPay» – два найпопулярніших сервіси по оплаті товарів чи послуг в Україні від двох банків – «ПриватБанк» та «monobank». Вибір завжди є приємним бонусом для користувачів, також присутній накладений платіж.

Із мінусів, на сайті відсутня гнучка система фільтрів, тому для кінцевого користувача це ускладнює вибір товару для покупки, особливо при великому асортименті. Покупцям доводиться витрачати більше часу на перегляд сторінок та з’являється ймовірність, що покупець не знайде потрібний товар.

На сайті присутні лише позитивні відгуки, що може вказувати на їх жорстку модерацію. Це може знижувати довіру користувачів до товарів, оскільки вони не можуть дізнатись інформацію про негативні сторони продукту, який вони хочуть придбати.

Також на сайті присутня інтеграція лише з «Новою Поштою», що може ускладнювати логістичні процеси у віддалені міста чи банально відлякувати користувачів, які не бажають користуватись саме цим сервісом доставки.

На рисунку 1.5 зображена сторінка з каталогом товарів «IGadget». На цьому сайті присутні різноманітні сучасні гаджети та побутова техніка, в тому числі і лампи.

Для кожного товару присутня детальна характеристика з інформацією про основні можливості товару. Інтерфейс світлий, приємний, але без підтримки мобільного інтерфейсу (рисунок 1.6, запускається десктопний інтерфейс).

Рисунок 1.5 – Сторінка з каталогом IGadget

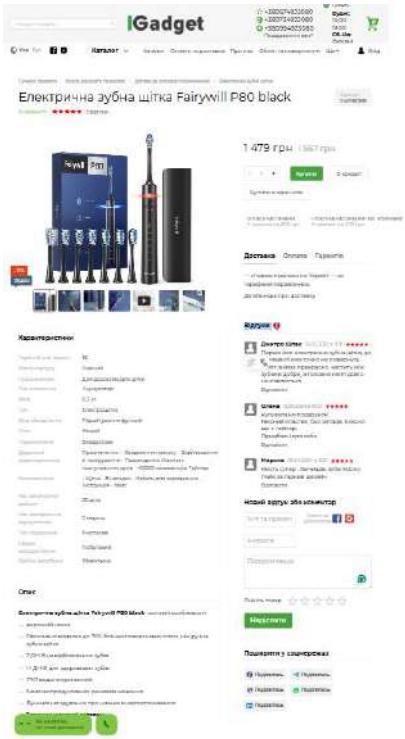


Рисунок 1.6 – Мобільний інтерфейс сайту

На сайті присутня розвинута система відгуків для кожного товару, реєстрація для написання відгуку не обов'язкова, достатньо написати власне

ім’я, прізвище та електронну пошту. Доступна функція поширити товар, як який допис у соцмережах, якщо він сподобався клієнту.

Реєстрація та вхід у власний акаунт присутня, але не є обов’язковою.

Порівняльна характеристика кожного із рішень зображена у таблиці 1.1

Таблиця 1.1 – Порівняльна таблиця рішень

Критерій	Intellect	Rozetka
Асортимент	Вузькоспеціалізований – лише лампи для дому/офісу	Дуже широкий – товари на будь-який смак і потребу
Технічні характеристики товару	Детальні (цоколь, потужність, температура світла, IP тощо)	Можуть бути обмежені для ламп, часто відсутні вузькі параметри
Інтерфейс	Мінімалістичний, без реклами – нічого не відволікає	Інтуїтивно зрозумілий, але з великою кількістю блоків, які можуть відволікати
Система фільтрів	Відсутня або мінімальна – ускладнює вибір при великому асортименті	Доволі потужна – дозволяє швидко знайти потрібний товар
Оплата	MonoPay, LiqPay, накладений платіж	Різноманітні способи оплати (картки, готівка, розстрочка тощо)
Доставка	Лише Нова Пошта	Укрпошта, Нова Пошта, Meest, власні пункти видачі
Мобільна адаптація	Є, зручна для смартфонів	Є

Продовження таблиці 1.1

Критерій	Intellect	Rozetka
Фокус на освітлення	Повністю спеціалізований на лампах	Освітлення – лише одна з численних категорій
Система відгуків	Присутні лише позитивні, можлива жорстка модерація	Повна – як позитивні, так і негативні, допомагає краще оцінити товар
Швидкість сайту	Вища через відсутність реклами	Може бути трохи нижча через кількість контенту
Обов’язкова реєстрація	Необов’язкова	Обов’язкова для покупки

1.3 Визначення функціональних та нефункціональних вимог до програмного забезпечення

Згідно з попереднім пунктом, вебзастосунок інтернет-магазину ламп повинен забезпечувати базовий набір функціоналу для зручної взаємодії з користувачем.

Було встановлено основні потреби користувачів при виборі ламп для освітлення.

Основний функціонал, який потрібно реалізувати:

– перегляд товарів – користувач повинен мати можливість переглядати каталог ламп, переглядати детальну інформацію про кожен товар, включаючи технічні характеристики (тип цоколя, потужність, температура світла, напруга, IP-рейтинг тощо);

– пошук та сортування – потрібно забезпечити зручний інтерфейс для пошуку товарів за ключовими словами, можливість сортування за ціною, популярністю, новизною та іншими критеріями;

– формування кошика – покупець повинен мати змогу додавати товари до кошика без необхідності створення облікового запису, редагувати вміст кошика та змінювати кількість товарів;

– оформлення замовлення – повинна бути наявна проста форма для оформлення замовлення з введенням необхідної інформації про покупця (адреса доставки, номер телефону тощо) та вибором способу оплати й доставки;

– підтримка мобільних пристроїв – інтерфейс повинен бути адаптований для комфортного перегляду сторінки на смартфонах через їх популярність.

Ідентифікація користувача буде відбуватись за допомогою генерації токена на стороні сервера по відбитку браузера. Відбиток браузера – технологія, що збирає інформацію, що стосується операційної системи користувача, типу браузера, клавіатури та інших характеристик. Обробляючи ці дані, створюється унікальний ідентифікатор, або «цифровий відбиток», для кожного користувача. Цей ідентифікатор залишається сталим в різних сеансах перегляду, що робить його надійним інструментом для ідентифікації відвідувачів за межами традиційних файлів cookie [4]. Коли новий користувач вперше заходить на сайт, серверна частина генеруватиме унікальний токен за цим принципом. Цей токен зберігається в локальному сховищі браузера користувача. Цей токен включає в себе різні характеристики браузера та пристрою (версія браузера, операційна система, встановлені плагіни, розмір екрана тощо).

При кожному наступному відвідуванні сайту з того самого браузера, вебзастосунок перевірятиме наявність раніше збереженого токена в локальному сховищі браузера. Якщо токен знайдено, система ідентифікує користувача як такого, що вже відвідував сайт раніше і підзавантажує інформацію про його кошик.

Таким чином, можна реалізувати не лише показ вмісту кошику, а і товари, які користувач переглядав раніше чи будь-які інші налаштування, наприклад історію пошуку.

Цей метод ідентифікації є специфічним для конкретного браузера та пристрою. Якщо користувач зайде на сайт з іншого браузера або пристрою, система розцінить це як нове відвідування і створить новий токен. Також Оскільки немає традиційної авторизації, користувач не може увійти у свій акаунт у звичному розумінні цього слова з різних пристроїв або браузерів.

Таким чином, кожен браузер матиме свій окремий акаунт, ідентифікований унікальним токеном.

Такий тип ідентифікації не повинен створити проблем для зручності, оскільки дані про користувача будуть постійно вводитись з нуля, а чутлива інформація (така як адреса доставки) передаватись за цим токеном не буде.

1.4 Висновки. Постановка задачі

Протягом виконання першого розділу дипломного проекту було детально проведено аналіз предметної області та її особливостей. Було здійснено дослідження інтернет-магазинів, які спеціалізуються на освітлювальних лампах та в цілому на освітлювальній техніці. Це допомогло глибше зрозуміти структуру таких платформ, та було виявлено ключові потреби користувачів таких сайтів.

Було також проведено аналіз вже існуючих рішень на основі популярних інтернет-магазинів та маркетплейсів, що пропонують подібну техніку. На основі аналізу було виявлено ключові характеристики, позитивні моменти та недоліки різних кінцевих продуктів по продажу ламп. Було враховано типові недоліки кожного сервісу.

За допомогою цієї інформації визначено основні вимоги до розробки програмного продукту, його ключовий функціонал та вимоги до інтерфейсу користувача. Ключові завдання на реалізацію такі як:

– розробка каталогу товарів з можливістю перегляду детальної інформації про кожну лампу;

– функціонал пошуку, фільтрації та сортування товарів за різними критеріями;

– розробка функціоналу кошика для формування замовлення без авторизації;

– добавлення простої форми для оформлення замовлення;

– адаптивність інтерфейсу для різних типів пристроїв.

2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕНЯ

2.1 Аналіз та проектування архітектури системи та вебдодатку

Було проведено аналіз ключових вимог до вебдодатку. Основні моменти, які були враховані при виборі архітектури, були:

- функціональність та кількість функцій;
- масштабованість;
- продуктивність;
- безпека;
- зручність розробки та підтримки.

Розглянуто декілька популярних архітектурних стилів: монолітна, мікросервісна архітектури та SPA.

Монолітна архітектура – усі компоненти вебдодатку (frontend, backend, база даних) розгорнуті як єдине ціле. Підходить для невеликих та середніх проектів з відносно простою функціональністю.

Мікросервісна архітектура – вебзастосунок розбивається на невеликі незалежні один від одного сервіси, кожен з яких відповідає за певну функціональність та може бути розгорнутий окремо. Краще підходить для великих та складних проектів.

Односторінковий додаток (SPA) – Frontend-частина розробляється як окремий застосунок, а з backend-частиною обмінюється даними через API.

Для реалізації цього проекту буде використовуватись патерн MVCS з SPA. MVCS розділює код між різними частинами вебзастосунку, тим самим покращує його структуру та полегшує масштабованість. Патерн MVCS складається з чотирьох компонентів: Model, View, Controller, та Service. На рисуну 2.1 схематично зображено спілкування між частинами застосунку.

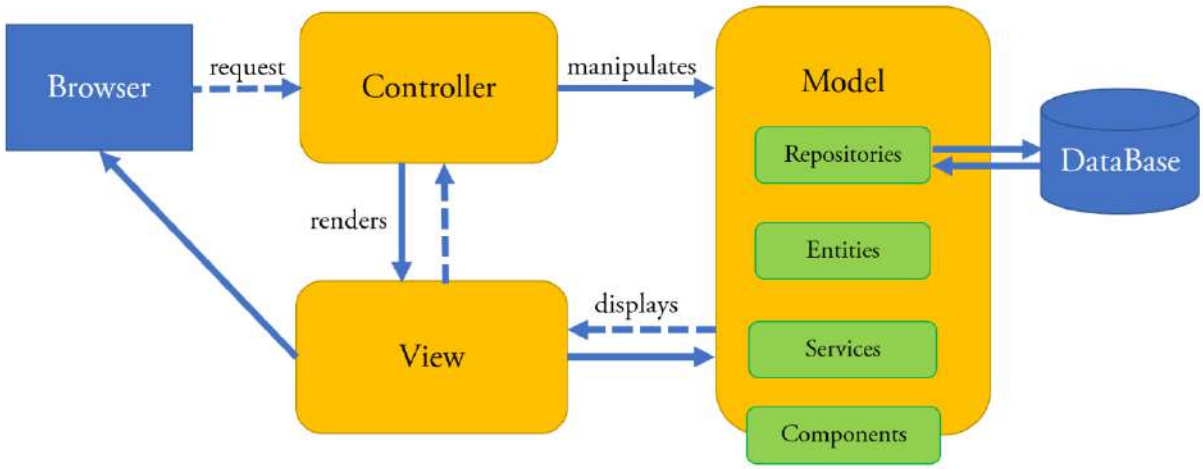


Рисунок 2.1 – Схематичне зображення взаємодії MVCS

Модель (Model) – керує та зберігає інформацію застосунку. Це, наприклад, робота з БД, файлами, зовнішніми API або будь-якими іншими джерелами даних. Модель може містити валідацію даних та логіку їх обробки. Наприклад, якщо потрібно перевірити чи є формат електронної пошти правильним перед збереженням, це може бути написано в моделі.

У моделі не повинно бути інформації про те, як її дані будуть відображатися користувачеві. Тим самим, це дозволяє легше змінювати інтерфейс та не модифіковувати модель.

Представлення (View) – відповідає за відображення інформації користувачеві. У вебзастосунках це, як правило, HTML-сторінки або компоненти інтерфейсу користувача. Представлення отримує дані від Контролера і форматує їх у зрозумілий вигляд для відображення користувачеві. У Представленні може бути мінімальна логіка, пов'язана безпосередньо з відображенням даних, наприклад, форматування дати. Основна бізнес-логіка все одно повинна знаходитися в інших компонентах.

Контролер (Controller) – отримує вхідні дані від користувача (наприклад, через форму на вебсторінці, URL-параметри або API-запити). На основі отриманих даних, Контролер взаємодіє з однією або кількома Моделями для отримання, оновлення або видалення даних. Після обробки

даних, контролер вирішує, яке Представлення потрібно відобразити користувачеві. Він може передавати дані з Моделі до Представлення.

Контролер повинен бути «тонким» і не містити складної бізнес-логіки. Його основна функція полягає в координації дій між Моделлю та Представленням.

Сервіс (Service) – відповідає за інкапсуляцію складної бізнес-логіки застосунку. Це дозволяє відокремити цю логіку від Контролерів та Моделей, що робить код більш організованим та зрозумілим. Контролери можуть викликати методи сервісних класів для виконання певних бізнес-операцій. Тим більше, Бізнес-логіка, реалізована в сервісах, може бути повторно використана в різних Контролерах.

Винесення бізнес-логіки в сервіси робить Контролери більш простими та зосередженими на отриманні вхідних даних та виборі представлення.

Переваги використання MVCS:

- спрощує розробку та розуміння коду;
- ліпше структуровує проект для розуміння;
- логіка Моделях та Сервісах може бути використана повторно;
- кожен компонент може бути протестований незалежно один від одного;
- спрощується спільна розробка;
- полегшується масштабованість проекту.

2.3 Аналіз та вибір типу бази даних, проектування її структури

При розробці системи електронної комерції було прийнято рішення використовувати реляційну базу даних MySQL. Цей вибір обґрунтовано наступними факторами:

- продуктивність та оптимізація;

– оптимізатор запитів, що дозволяє ефективно обробляти складні запити;

– механізми кешування, які значно підвищують швидкість отримання даних;

– безпека даних;

– підтримка шифрування даних у спокої та під час передачі;

– гранульована система привілеїв для контролю доступу;

– підтримка SSL для захищених підключень;

– аудит операцій з базою даних;

– сумісність і підтримка;

– багата екосистема інструментів для адміністрування, резервного копіювання, моніторингу та реплікації (наприклад, MySQL Workbench, Percona Toolkit, phpMyAdmin);

– активна спільнота користувачів і регулярні оновлення безпеки та продуктивності від Oracle;

– підтримка масштабування завдяки механізмам реплікації та можливості горизонтального розподілу навантаження (шардінг);

– низькі вимоги до апаратного забезпечення порівняно з іншими СУБД корпоративного рівня (наприклад, Oracle DB, Microsoft SQL Server);

– наявність безкоштовної версії – MySQL Community Edition, яка надає всі базові функції без ліцензійних витрат;

– перевірена надійність і стабільність у реальних проєктах з великим обсягом транзакцій;

– добре документований інтерфейс і велика база знань для швидкого вирішення типових проблем.

На рисунку 2.2 зображено схему бази даних, на рисунку 2.3 – продовження схеми, де відображається ще поля для секції частих запитань від користувачів

Рисунок 2.2 – Схема бази даних

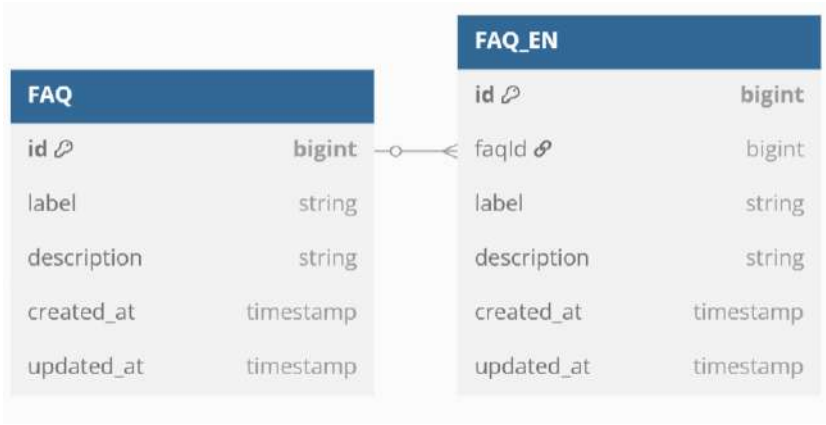


Рисунок 2.3 – Схема таблиць частих запитань з підтримкою багатомовності

Основні сутності бази даних:

- User – інформація про користувача системи;
- Category – категорія товару;
- Product – товар з детальною інформацією;
- Order – замовлення користувача;
- Cart – кошик користувача;
- Media – медіафайли (зображення та відео товарів);
- FAQ – часті запитання.

Зв'язуючі таблиці:

- order_product – зв'язок між замовленнями та товарами
- cart_item – товари в кошику
- similar_product – зв'язок між схожими товарами
- product_en – локалізація товарів англійською мовою
- faq_en – локалізація FAQ англійською мовою

Таблиця «User» – розширена стандартна таблиця користувачів Laravel для підтримки токенів автентифікації. Змінено обов'язковість полів email і password для підтримки автентифікації за допомогою токену. Поля email і password збереглись для підтримки автентифікації адміністратора у свій

акаунт. Особисті дані користувача (ім'я, прізвище, телефон) видалені з таблиці і перенесені в інформацію про замовлення.

Таблиця «Category» – має зв'язок з магазинами (shopId), що дозволяє створювати мультивендорну систему. Містить описи для відображення на сторінках категорій.

Таблиця «Product» – центральна таблиця системи, що зберігає основну інформацію про товари. Підтримує SEO-оптимізацію через slug-поля. Зберігає ціни, знижки, залишки на складі. Має зв'язок з категоріями. Розширена додатковими полями для короткого опису та технічних специфікацій

Таблиці локалізації («Product_en», «Faq_en») – реалізовано підхід з окремими таблицями для перекладів. Зв'язок з основною таблицею через зовнішній ключ. Всі поля перекладів зроблені необов'язковими для підтримки часткових перекладів.

Таблиці замовлень («Order», «Order_product») – структура з головною таблицею замовлень та підпорядкованою таблицею позицій замовлення. Збереження статусу замовлення в окремому полі. Включення інформації для доставки («Нова Пошта») - місто, відділення для інтеграції API на наступнихх етапах. Зберігається контактна інформація замовника безпосередньо в замовленні, а не в профілі користувача

Таблиці кошика («Cart», «Cart_item») – окрема таблиця для кошиків з можливістю зміни статусу Підпорядкована таблиця з елементами кошика. Зберігається ціна товару в момент додавання до кошика для захисту від зміни ціни.

Таблиця «Similar_product» – реалізовано через зв'язуючу таблицю для підтримки рекомендацій схожих товарів. Відношення багато-до-багатьох між товарами.

Таблиця «Media» – створюється за допомогою Spatie Media Library, отож використовує його структуру для управління медіафайлами. Забезпечує

поліморфні зв'язки з різними моделями. Підтримує зберігання конвертації зображень, респонсивні версії та метадані.

2.3 Проектування інтерфейсу користувача

Вебзастосунок буде виконаний у мінімалістичному стилі з використанням чорних та темно-зелених кольорів. На рисунку 2.4 зображено секцію головної сторінки з товарами. Натиснувши картку з товаром, відбувається перехід на сторінку з його повною інформацією. Ця сторінка зображена на рисунку 2.6

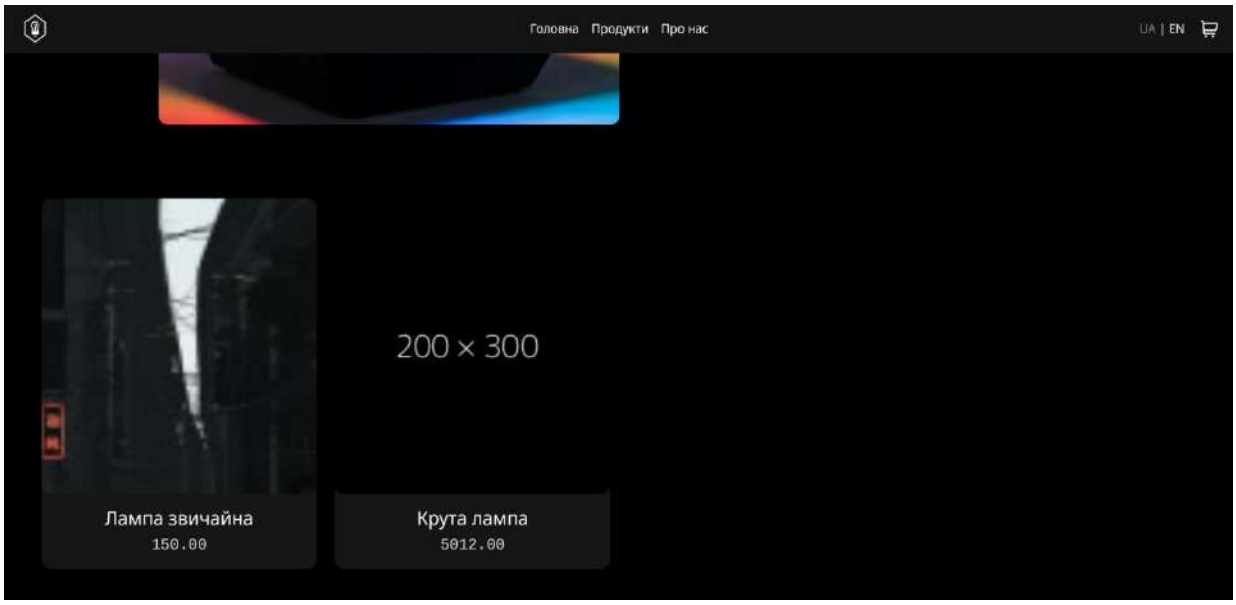


Рисунок 2.4 – Секція з товарами

На головній сторінці також можна побачити hero-розділ [5] – область, яка одразу з'являється на екрані під логотипом і меню. В ідеалі ця частина сторінки повинна містити інформацію про чотири речі:

- що проект можете запропонувати;
- чому люди повинні довіряти цьому проекту;
- переваги саме цього проекту;
- які дії користувачі повинні зробити далі.

На рисунку 2.5 зображено hero-розділ головної сторінки.

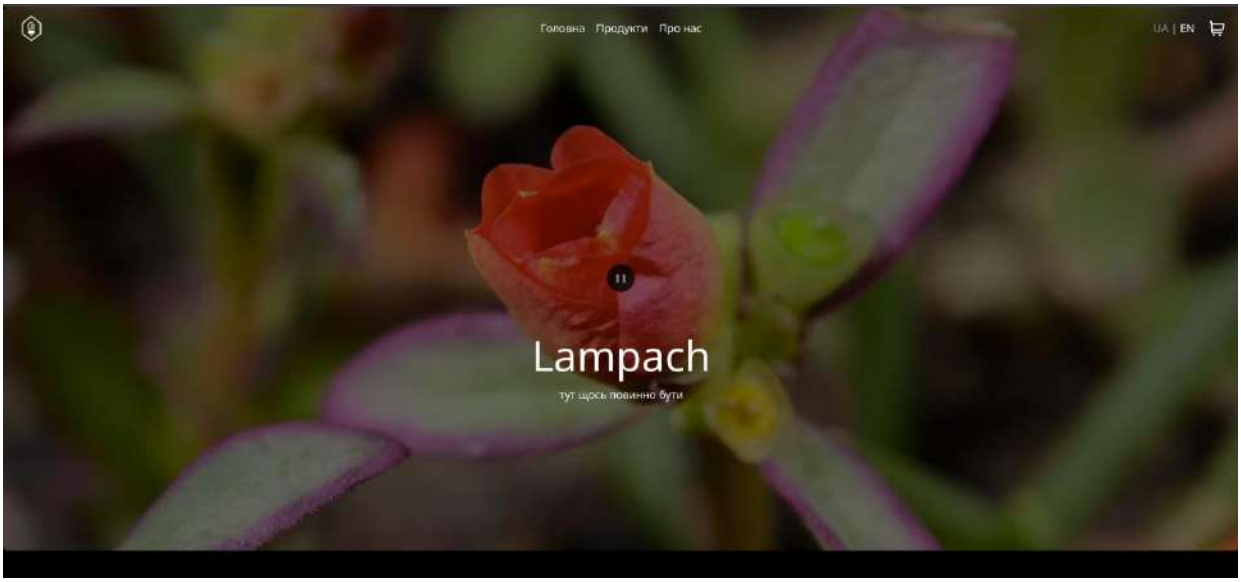


Рисунок 2.7 – Hero-секція

Також на головній сторінці присутня секція з частими запитаннями, що зображено на рисунку 2.6.

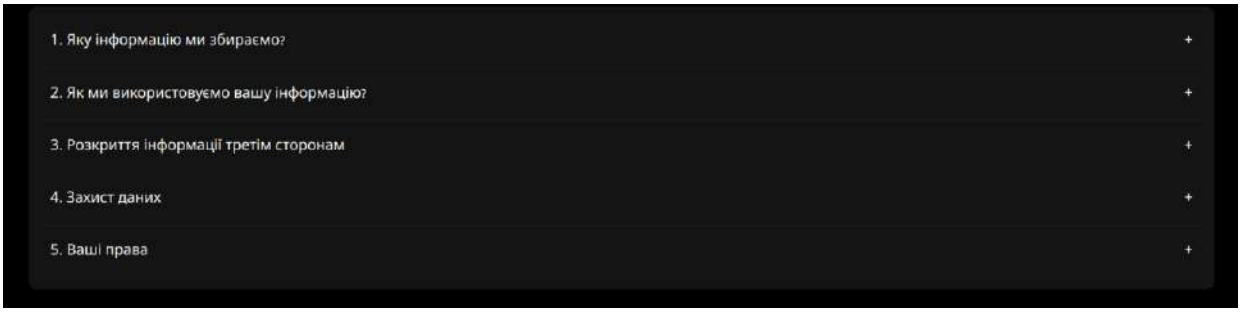


Рисунок 2.6 – Секція з частими запитаннями

На рисунку 2.7 зображена сторінка з повною інформацією про товар.

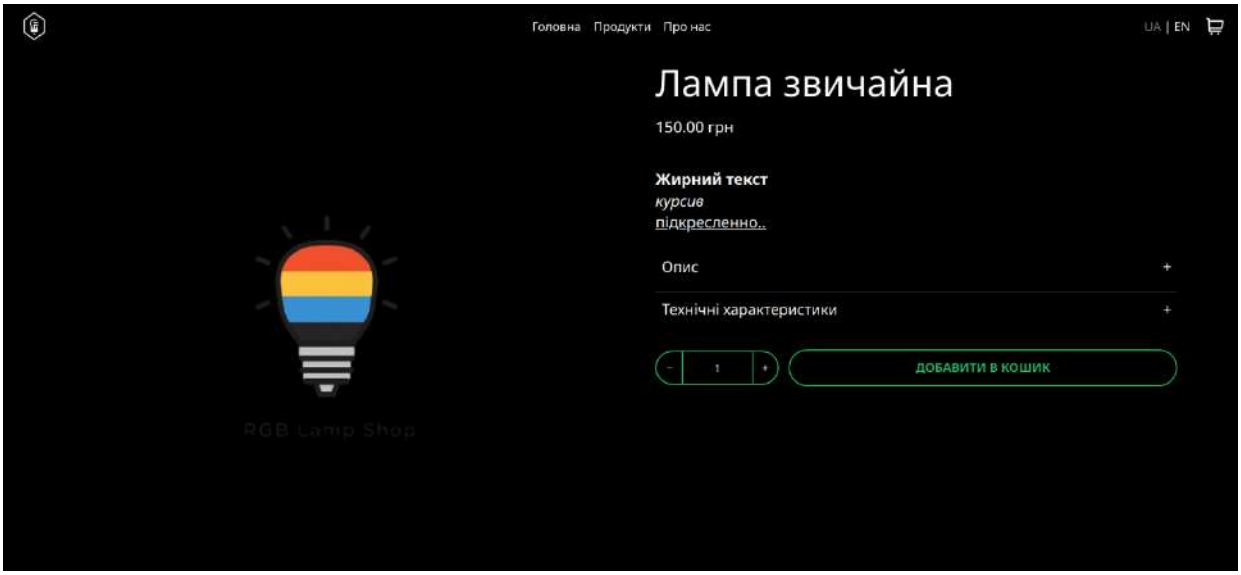


Рисунок 2.7 – Сторінка з повною інформацією про товар

На ній зображено: два спадних меню з описом лампи та технічними характеристиками, лічильник з кількістю товару, кнопка з добавленням товару в кошик. Після кліку на кнопку «Добавити в кошик», користувач перейде на сторінку з кошиком, що зображений на рисунку 2.8.

Тут можна залишити контакту інформацію про замовника, побачити загальну ціну замовлення, змінити своє замовлення остаточно (добавити більшу кількість одного товару, чи взагалі його видалити). Адреса доставки буде підтягувати адреси поштоматів та відділень Нової Пошти за допомогою API.

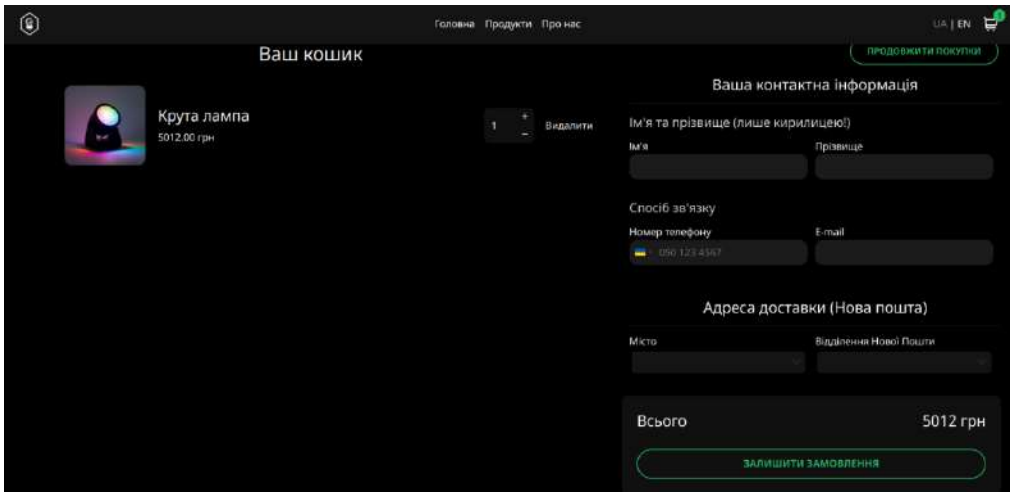


Рисунок 2.8 – Сторінка з кошиком

Після підтвердження замовлення, буде надсилатись повідомлення у месенджер Telegram адміністратору. На рисунку 2.9 зображено приклад такого повідомлення.

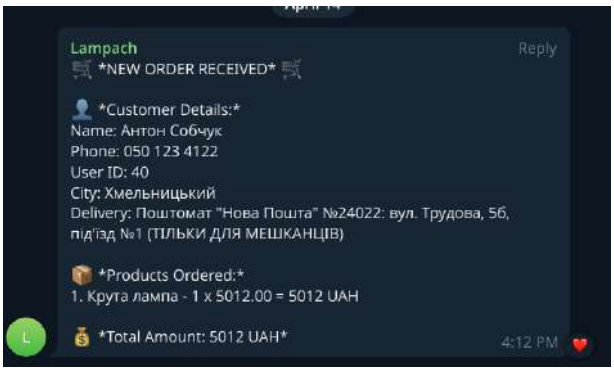


Рисунок 2.9 – Повідомлення про нове замовлення у Telegram

Для реалізації багатомовності, у адміністративній панелі Filament буде реалізовано два вибори між мовами: українська та англійська. На рисунках 2.10 та 2.11 відображено, як це буде виглядати при додаванні даних у таблицю з частими запитаннями та продуктами.

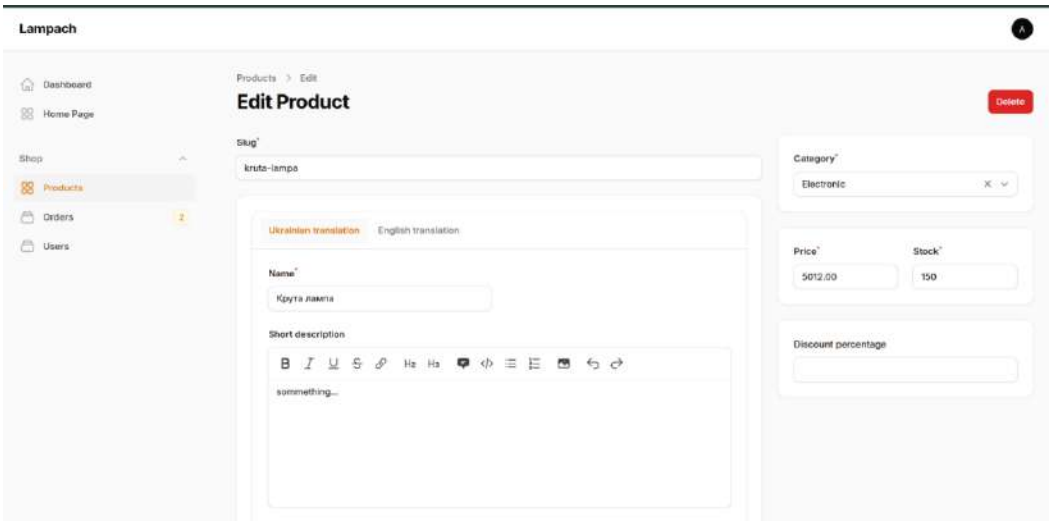


Рисунок 2.9 – Створення запису продукту

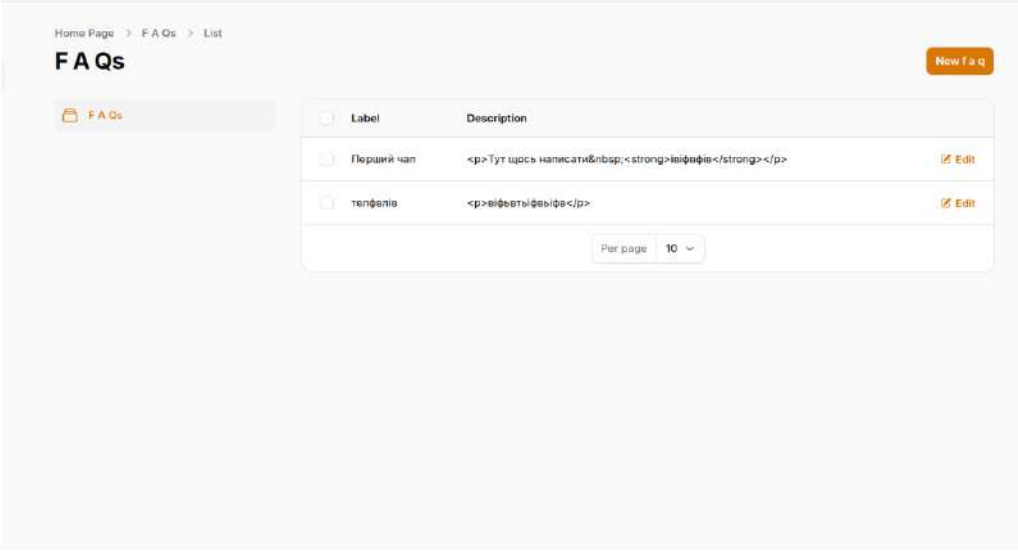


Рисунок 2.10 – Вид частих запитань у адміністративній панелі

2.4 Аналіз та вибір технологій і методів реалізації системи

Бізнес-логіка – це користувацькі правила або алгоритми, які керують обміном інформацією між базою даних та користувацьким інтерфейсом. [6]

Вебзастосунок складатиметься з такої архітектури: backend та frontend.

Frontend частина виступатиме в ролі клієнтського інтерфейсу та включатиме такі компоненти:

– Vue.js буде використовуватись як основний JavaScript фреймворк. З його допомогою можна створити користувацький інтерфейс з високою продуктивністю. Це сучасний фреймворк, у якого є гарна документація та доволі низький поріг входу;

– Vue Router буде інтегровано для реалізації Single Page Application (SPA) архітектури, яка забезпечить плавні переходи між сторінками без підвантаження всієї сторінки;

– Tailwind CSS використовуватиметься як CSS фреймворк, що значно прискорить процес розробки інтерфейсу. Цей інструмент дозволить ефективно стилізувати компоненти без необхідності писати власний CSS з нуля;

– додаткові Vue.js бібліотеки будуть використані для покращення користувацького інтерфейсу, додаючи додаткову функціональність;

– спілкування з backend буде відбуватись за допомого REST API з використанням JSON як формату обміну даними.

Основою backend стане PHP та його фреймворк Laravel. Він надасть стабільну та безпечну роботу серверної логіки. Laravel пропонує багатий набір готових рішень для аутентифікації, маршрутизації, кешування та обробки запитів.

Однією з ключових переваг Laravel є його елегантний синтаксис та читабельність коду, що полегшує розробку та підтримку. Фреймворк містить потужний ORM Eloquent, який спрощує взаємодію з базами даних, а також зручні механізми для керування структурою бази даних за допомогою міграцій. Крім того, Laravel має велику та активну спільноту розробників, тобто має високий рівент підтримки та доступ до численних пакетів і ресурсів.

Обробка запитів від frontend частини буде здійснюватися через RESTful API контролери, які забезпечать валідацію вхідних даних, авторизацію користувачів та виконання відповідних бізнес-операцій. API контролери також включатимуть механізми обробки помилок з поверненням стандартизованих HTTP-статусів та повідомлень

Spatie Media Library буде інтегровано для оптимізованого управління медіафайлами, включаючи завантаження, зберігання, трансформацію та доставку зображень та інших медіа-ресурсів. Ця бібліотека спрощує роботу з медіафайлами та забезпечує їх оптимізацію. Бібліотека підтримує широкий спектр файлів, включаючи зображення (jpg, png, svg, webp, avif), відео (mp4, mov, webm) та PDF-файли, та дозволяє автоматично створювати їх мініатюри для оптимізації завантаження сторінки. Spatie Media Library також надає можливості для різноманітних трансформацій зображень, як-от зміна розміру, обрізка, застосування фільтрів тощо. Для забезпечення конфіденційності передбачена можливість роботи з приватним сховищем, включаючи генерацію тимчасових URL-адрес для доступу до файлів.

Filament буде використано для швидкого створення адміністративного інтерфейсу з багатими функціональними можливостями. Цей CRUD-генератор дозволить автоматизувати створення форм, таблиць та інших елементів адміністративної панелі, значно скорочуючи час розробки та зусилля, необхідні для створення повноцінного адмін-інтерфейсу. Filament побудований на основі Livewire та Blade, що забезпечує швидку розробку та гнучкість у налаштуванні інтерфейсу, особливо на PHP з Laravel, які мають тісну інтеграцію з цими компонентами. Завдяки вбудованій підтримці Spatie Media Library, керування медіафайлами стає простим. Крім того, Filament пропонує потужну систему розширення, дозволяючи додавати користувацькі компоненти та функції, а також має вбудовані можливості для керування користувачами, ролями та правами доступу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА РОЗРОБКА

3.1 Розробка бази даних

Для забезпечення роботи застосунку з базою даних використовується зв'язка технологій MySQL, Docker та конфігураційні можливості фреймворку Laravel.

1. Конфігурація підключення в Laravel (.env)

Файл .env використовується для зберігання чутливих даних та специфічних для середовища налаштувань, зокрема параметрів підключення до бази даних. Наприклад, конфігурація для БД:

```
DB_CONNECTION=mysql
DB_HOST=mysql
DB_PORT=3306
DB_DATABASE=lampach
DB_USERNAME=root
DB_PASSWORD=
```

– DB_CONNECTION=mysql: вказує, що використовується драйвер бази даних MySQL;

– DB_HOST=mysql: визначає хост, де розташована база даних. У даному випадку mysql є ім'ям сервісу Docker, що дозволяє контейнеру застосунку знаходити контейнер бази даних за його іменем у його спільній мережі;

– DB_PORT=3306: стандартний порт для підключення до MySQL;

– DB_DATABASE=lampach: ім'я бази даних, до якої підключається застосунок;

– DB_USERNAME=root: ім'я користувача бази даних.

– DB_PASSWORD=: Пароль користувача бази даних. В наданому фрагменті пароль для користувача root порожній. Для робочого середовища настійно рекомендується встановлювати надійний пароль.

Ці змінні зчитуються Laravel під час завантаження застосунку і використовуються для встановлення з'єднання з базою даних.

2. Налаштування сервісу бази даних за допомогою Docker Compose (docker-compose.yml)

Файл docker-compose.yml описує середовище Docker, дозволяючи легко підняти всі необхідні сервіси (вебсервер, база даних тощо) однією командою. Налаштування БД тут виглядає так:

```
mysql:
  image: mysql:8.0
  container_name: mysql
  restart: unless-stopped
  environment:
    MYSQL_DATABASE: lampach
    MYSQL_USER: root
    MYSQL_PASSWORD: 1234
  ports:
    - "3344:3306"
  volumes:
    - mysql_data:/var/lib/mysql
  networks:
    - laravel
```

mysql: визначає ім'я сервісу Docker. Це ім'я (mysql) використовується в файлі .env як DB_HOST.

image: mysql:8.0: вказує, який образ Docker використовувати для створення контейнера. Тут використовується офіційний образ MySQL версії 8.0.

container_name: mysql: призначає конкретне ім'я контейнеру.

restart: unless-stopped: налаштовує політику перезапуску контейнера. Він буде автоматично перезапускатись, якщо зупиниться з помилкою, за винятком випадків, коли його зупинено вручну.

environment: секція для встановлення змінних оточення всередині контейнера MySQL.

MYSQL_DATABASE: lampach: створює базу даних з іменем lampach при першому запуску контейнера.

MYSQL_USER: root: встановлює ім'я користувача (хоча root вже існує, це може бути використано для інших цілей або якщо не встановлено пароль root напряму).

MYSQL_PASSWORD: 1234: Встановлює пароль для користувача root. Важливо, щоб пароль був одним і тим же, що і в .env

ports: - "3344:3306": прокидає порт 3306 контейнера (де працює MySQL) на порт 3344 на хост-машині. Це дозволяє отримати доступ до бази даних ззовні контейнера (наприклад, за допомогою PHPStorm або MySQL Workbench) за адресою localhost:3344.

volumes: - mysql_data:/var/lib/mysql: використовує іменований том mysql_data для зберігання даних бази даних (/var/lib/mysql всередині контейнера). Це означає, що дані бази даних будуть збережені навіть після зупинки, видалення або оновлення контейнера.

networks: - laravel: підключає сервіс MySQL до мережі Docker з іменем laravel, що дозволяє іншим сервісам у цій же мережі спілкуватися з базою даних за її сервісним іменем (mysql).

За допомогою вбудованих інструментів PHPStorm було реалізовано ефективну роботу з базою даних MySQL, що функціонує в контейнері Docker. Ключові моменти:

– підключення та моніторинг бази даних – зручний доступ до структури бази даних та її вмісту безпосередньо з IDE;

– перегляд та навігація схемою даних через вбудований інструмент Database Tool Window;

– виконання SQL-запитів;

– перегляд та редагування даних у таблицях – була можливість переглядати вміст таблиць у зручному табличному вигляді та за необхідності швидко редагувати дані;

– інтеграція з командами Artisan – дозволило зручно виконувати команди, пов'язані з базою даних, такі як php artisan migrate для застосування міграцій та php artisan db:seed для заповнення бази даних початковими даними, безпосередньо з терміналу IDE.

Щоб налаштувати підключення БД в PHP Storm, потрібно:

1. у вікні Database натиснути кнопку «+», знайти і обрати MySQL (рисунок 3.1)

2. правильно налаштувати дані, взяти їх із docker-compose (рисунок 3.2). Перевірку правильно підключення можна здійснювати через Test Connection

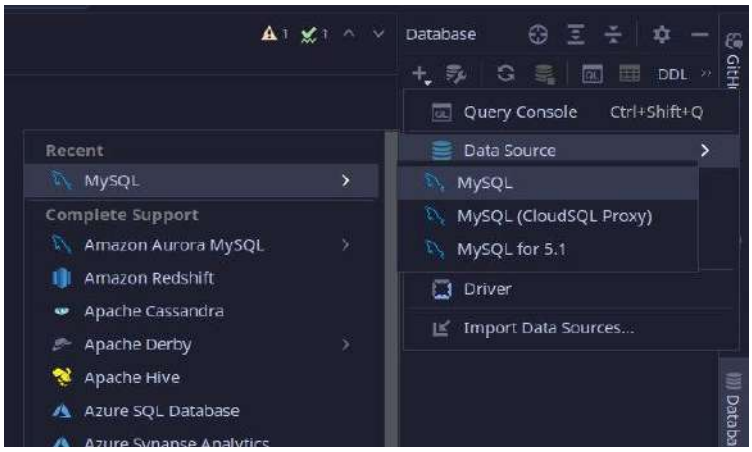


Рисунок 3.1 – Вибір драйвера

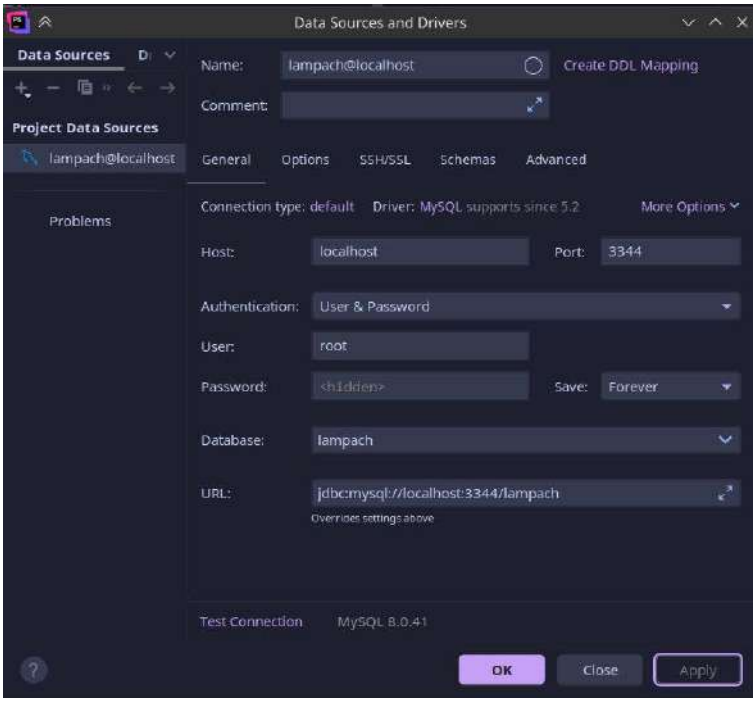


Рисунок 3.2 – Налаштування підключення

3.2 Розробка програмних модулів

Проєкт складається з кількох основних модулів/частин, кожна з яких відповідає за окрему функціональність. Загальна структура зображена на рисунку 3.3

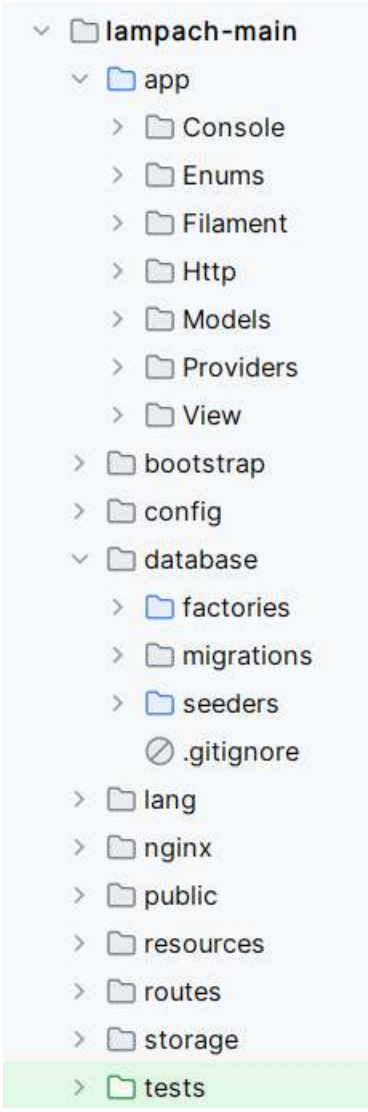


Рисунок 3.3 – Структура проекту

Frontend частина знаходиться у /resources/. Для ввімкнення роботи vue.js з Laravel було добавлено в основний blade-компонент app.blade.php наступне:

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>Lampach</title>

        @vite(['resources/css/app.css', "resources/js/app.js"])
    </head>
    <body class="font-montserrat antialiased">
        <div id="app">

        </div>
    </body>
</html>
```

Тобто, він підзавантажує основний JS файл (app.js), в якому наявна імплементація роботи Vue.js та Vue router, а також основний файл, який відповідає за коректну роботу Tailwind CSS.

У свою чергу, у файлі app.js також знаходиться реалізація багатомовності вебзастосунку. Код реалізації наведено нище:

```
app.use(i18nVue, {
    resolve: async (lang) => {
        const savedLang = localStorage.getItem('lang');
        const currentLang = savedLang || lang || 'uk';
        const langs = import.meta.glob('../../lang/*.json');
        const languageModule = langs[`../../lang/${currentLang}.json`];

        if (!languageModule) {
            return await langs[`../../lang/uk.json`]();
        }

        return await languageModule();
    }
});
```

Vue Router налаштовується в окремому файлі

Багатомовність реалізована за допомогої бібліотеки Laravel Vue i18n. Принцип його роботи такий: беруться файли локалізації, які знаходяться в місці по замовчуванню у Laravel (файли /lang/{код_мови}/*.php), генерує з них JSON-масив, та ці JSON-масиви використовує у файлах Vue.js. Відбувається це все автоматично при запуску команди npm run dev. Звернення відбувається за допомогою команди:

```
trans(' ключ_перекладу ')
```

Після цього, відбувається завантаження компонентів Vue.js, що знаходяться у resources/js/pages. Базовим компонентом є App.vue, у якому містяться такі компоненти, як «шапка» сторінки, тобто елементи, які будуть часто використовуватись на всіх сторінках, і щоб не завантажувати їх для кожної сторінки окремо, вони завантажуються зразу ж в основному компоненті.

Структура Vue.js компоненту виглядає наступним чином:

```
<template>
   <!—HTML код -->
</template>

<script>
```

```
<!-- JavaScript код -->
export default {
    data() {
      return {
          <!-- Основні константи або змінні компоненту -->
      }
    },
    methods: {
        <!-- методи, які використовуються -->
    },
    components: {
<!-- Компоненти, які використовуються у секції <template> та були імпортовані
-->
}
};
</script>

<style>
<!-- CSS стилі -->
</style>
```

Для стилізації використано Tailwind CSS, який працює за принципом класів. Це означає, що для застосування будь-якого стилю достатньо в HTML-компоненті додати відповідний клас, визначений у Tailwind CSS. Цей фреймворк надає величезний набір готових класів для вирішення різноманітних задач стилізації.

У сфері стилізації Tailwind CSS пропонує класи для керування практично всіма аспектами зовнішнього вигляду елементів. Серед них класи для налаштування кольору тексту (text-red-500, text-gray-900), фону (bg-blue-200, bg-white), розміру шрифту (text-sm, text-xl), міжрядкового інтервалу (leading-relaxed, leading-tight) та багато іншого. Комбінуючи ці класи, розробник може швидко і гнучко створювати складні візуальні рішення.

Для анімації Tailwind CSS надає класи для керування переходами та анімаціями. Класи, що починаються з префікса transition-, дозволяють визначати, які властивості підлягають анімації та з якою тривалістю (transition-all, transition-colors, duration-300). Крім того, фреймворк включає готові анімації, які можна застосувати за допомогою класів з префіксом animate- (наприклад, animate-spin для створення ефекту обертання, animate-bounce для підстрибування).

Tailwind CSS має потужну систему адаптивного дизайну, що базується на медіа-запитах. Застосовуючи префікси до класів, можна визначати стилі, які будуть активними лише на певних розмірах екранів. Основні префікси включають sm: (для екранів розміром 640px і більше), md: (768px і більше), lg: (1024px і більше), xl: (1280px і більше). Це дозволяє легко створювати інтерфейси, які коректно відображаються на пристроях з різною шириною екрана.

Серверна частина проєкту побудована на базі сучасного PHP-фреймворку Laravel з використанням додаткових інструментів та пакетів для розширення функціональності.

В якості основи серверної частини використовується фреймворк Laravel, який забезпечує структурований підхід до розробки вебзастосунків та надає широкий спектр вбудованих функцій. Laravel використовує архітектурний шаблон MVC (Model-View-Controller), що дозволяє ефективно розділити бізнес-логіку, представлення та обробку запитів.

Серед використаних компонентів Laravel:

- Eloquent ORM – для роботи з базою даних та створення моделей;
- Blade – як для головного файлу html;
- Artisan – консольний інструмент для автоматизації задач;
- міграції – для керування структурою бази даних.

Для побудови адміністративної панелі використовується пакет Filament, який надає стек інструментів (Tailwind CSS, Alpine.js, Laravel та Livewire) для швидкого створення адмін-панелей. Filament дозволяє зручно керувати даними, створювати форми, таблиці та інші елементи UI. Він є досить гнучким і дозволяє налаштовувати зовнішній вигляд, поведінку та функціональність відповідно до потреб вашого проекту. Приклад коду для ресурсу Filament:

```
#[AllowDynamicProperties] class OrderResource extends Resource
{
    protected static ?string $model = Order::class; //основна модель з якою відбуваються операції
```

```
    //Форма для редагування або створення запису
    public static function form(Form $form): Form {
        return $form
            ->schema([
                Group::make()->schema([
                    Forms\Components\Section::make()
                        ->schema([
                            Forms\Components\Select::make('userId')
                                ->required()
                                ->relationship('user', 'name')
                                ->searchable(),

                            Forms\Components\ToggleButtons::make('status')
                                ->inline()
                                ->options(OrderStatus::class)
                                ->required(),

                            TextInput::make('phoneNumber'),
                            TextInput::make('cityRef')->columnSpan(2),
                            TextInput::make('warehouseSettlementRef')-
>columnSpan(2),
                        ])->columns(2),
                        static::getItemsRepeater()->columns(2)-
>columnSpan(2),
                ])->columns(2)->columnSpan(['lg' => 2]),
            ]);
    }

//таблиця з даними
    public static function table(Table $table): Table {
        return $table
            ->columns([
                TextColumn::make('id')
                    ->label('ID'),
                TextColumn::make('user.name')
                    ->searchable()
                    ->sortable()
                    ->label('User'),
                TextColumn::make('status')
                    ->badge()
                    ->color(fn (string $state): string =>
OrderStatus::tryFrom($state)?->getColor() ?? 'gray')
                    ->icon(fn (string $state): ?string =>
OrderStatus::tryFrom($state)?->getIcon())
                    ->label('Status')
                    ->formatStateUsing(fn (string $state): ?string =>
OrderStatus::tryFrom($state)?->getLabel()),
                TextColumn::make('totalAmount')
                    ->numeric(decimalPlaces: 0)
                    ->label('Total Amount')
                    ->summarize([
                        Tables\Columns\Summarizers\Sum::make()]
                    ),
                TextColumn::make('created_at')
                    ->date()
                    ->sortable(),

            ])->defaultSort('created_at', 'desc')
            ->filters([
                //
```

```
            ])
            ->actions([
                Tables\Actions\EditAction::make(),
            ])
            ->bulkActions([
                Tables\Actions\BulkActionGroup::make([
                    Tables\Actions\DeleteBulkAction::make(),
                ]),
            ])->groups([
                Tables\Grouping\Group::make('created_at')
                    ->label('Order Date')
                    ->date()
                    ->collapsible(),
            ]);
    }

//сторінки та їх відповідні класи (для додаткового редагування, якщо не
вистачило функціоналу ресурсу)
    public static function getPages(): array {
        return [
            'index' => Pages\ListOrders::route('/'),
            'create' => Pages\CreateOrder::route('/create'),
            'edit' => Pages\EditOrder::route('/{record}/edit'),
        ];
    }

//віджет для відображення додаткової інформації
    public static function getWidgets(): array
    {
        return [
            OrderStats::class,
        ];
    }
}
```

Приклад відображення Filament зображено на рисунку 3.5

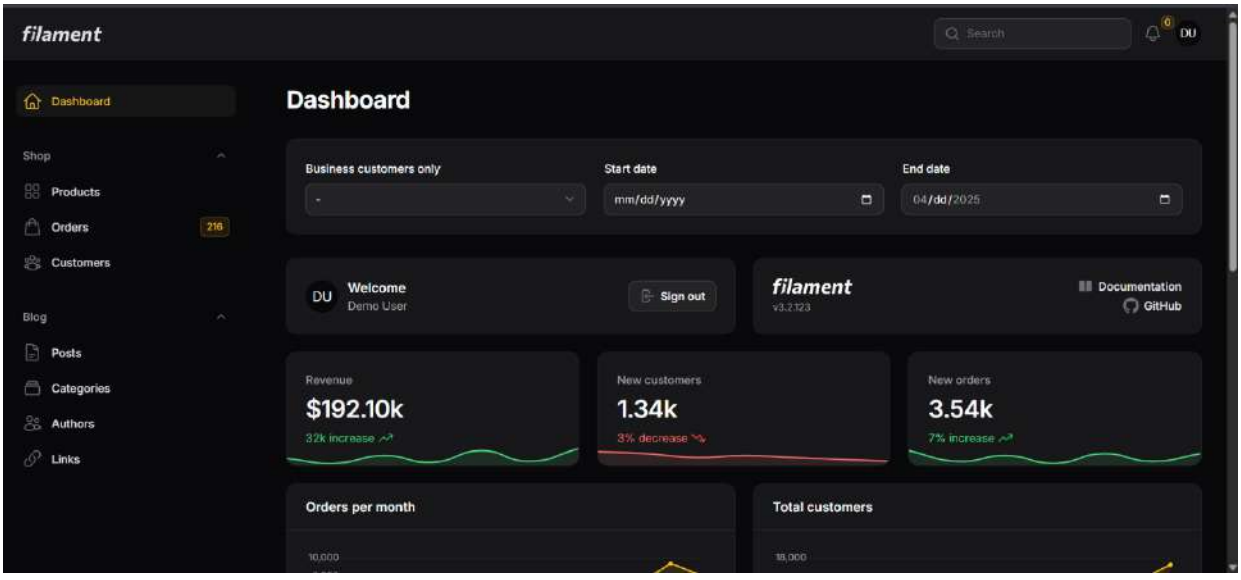


Рисунок 3.5 – Приклад дизайну Filament

Завдяки використанню Livewire, більшість дій в адміністративній панелі виконуються без повного перезавантаження сторінки, тим самим покращуючи користувацький досвід.

Для розширення функціональності Laravel та Filament використовуються пакети від Spatie: spatie/laravel-medialibrary - для керування медіафайлами. Пакет забезпечує зручний інтерфейс для прикріплення медіафайлів до моделей, генерації прев'ю та оптимізації зображень:

```
class Product extends Model implements HasMedia {
    use InteractsWithMedia;
}
```

Отримання URL зображень:

```
$thumbUrl = $product->getFirstMediaUrl('product_images', 'thumb');
```

Для керування залежностями проєкту використовується Composer - стандартний пакетний менеджер для PHP. Composer – це інструмент керування залежностями на рівні застосунку для мови програмування PHP. Його головна мета – спростити процес включення та оновлення зовнішніх бібліотек та пакетів, від яких залежить ваш проєкт. Composer не є менеджером пакетів на рівні системи (як, наприклад, apt чи yum у Linux), він працює в контексті окремого проєкту.

У файлі composer.json визначаються всі залежності проєкту та їх версії:

```
json{
    "require": {
        "php": "^8.1",
        "laravel/framework": "^10.0",
        "filament/filament": "^3.0",
        "spatie/laravel-permission": "^5.10",
        "spatie/laravel-medialibrary": "^10.0",
        "nutgram/nutgram": "^4.0",
        "daaner/novaposhta": "^1.0",
        "intervention/image": "^2.7"
    }
}
```

Composer зазвичай використовує Packagist (https://packagist.org/) як основний репозиторій, звідки завантажуються пакети. Однак, можна налаштувати й інші джерела, включаючи власні приватні репозиторії.

Для інтеграції з Telegram API використовується бібліотека Nutgram, яка дозволяє створювати Telegram-ботів з використанням сучасних підходів PHP 8. Вона буде використана, щоб відправляти повідомлення адміністратору про нове замовлення. Основні компоненти реалізації:

```
class TelegramController extends Controller
{
    protected $bot;

    public function __construct(Nutgram $bot) {
        $this->bot = $bot;
    }

    public function sendDataToTelegram(Request $request) {
        $data = $request->all();
        $message = "🛒 *NEW ORDER RECEIVED* 🛒\n";
        $chatId = config('services.telegram.chat_id');
        $this->bot->sendMessage($message, $chatId);
        return response()->json(['success' => true]);
    }
}
```

Щоб створити бота та отримати його botID потрібно:

1. Знайти бота BotFather в пошуку Telegram
2. Нажати команду /newbot (рисунок 3.6)
3. Дати назву боту
4. Дати юзернейм, який буде закінчуватись на bot
5. Отримати API-ключ (рисунок 3.7)

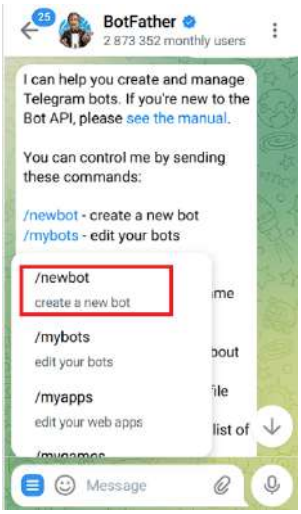


Рисунок 3.6 – Вибір команд

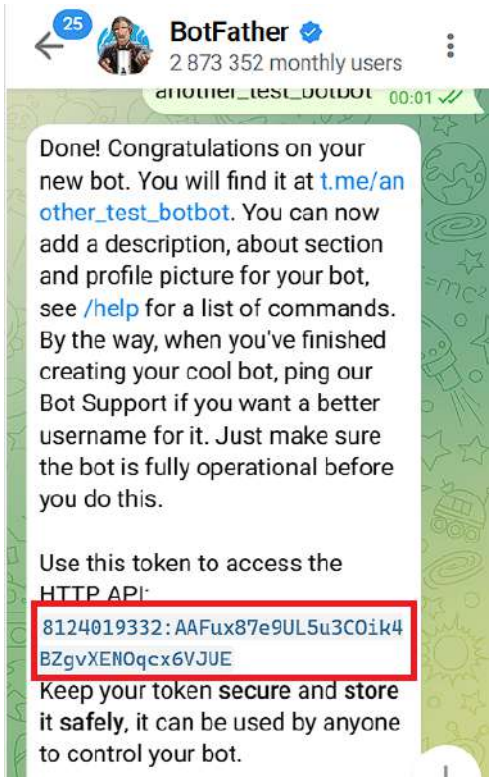


Рисунок 3.7 – Telegram-bot API-ключ

Щоб дізнатись власний chatID (щоб бот знав, куди відправляти повідомлення), можна використати додаткові боти, наприклад Creation Date Bot (рисунок 3.8)

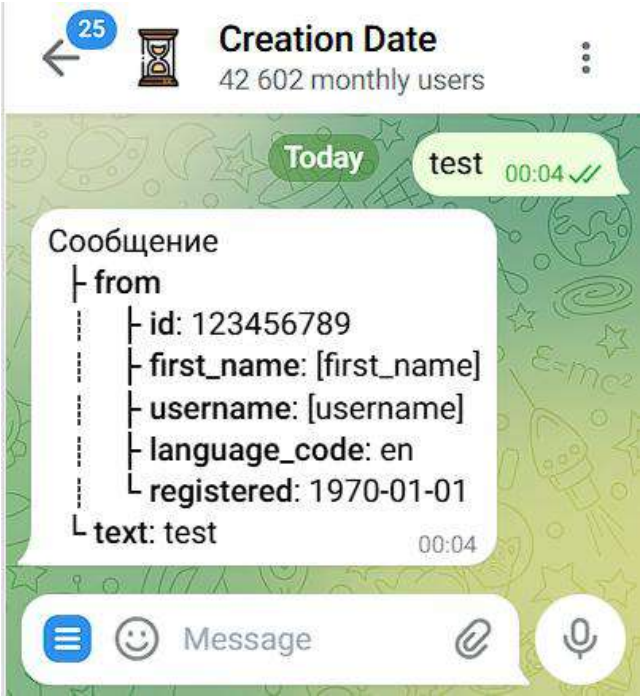


Рисунок 3.8 – Основна інформація про користувача, включно з його ID

Для взаємодії з API Нової Пошти використовується пакет daaner/novaposhta, який надає зручний доступ до всіх адресів та населенних пунктів:

```
class NovaPoshtaController extends Controller
{

    protected $adr;

    public function __construct(Address $adr) {
        $this->adr = $adr;
    }
//отримуються населенні пункти
    public function getCities(Request $request) {
        $city = $request->query('city');
        return $this->adr->getCities($city);
    }
//отримуються відділення
    public function getWarehouseSettlements(Request $request) {
        $city = $request->query('cityRef');
        return $this->adr->getWarehouses($city, false);    }
}
```

В цьому API існує обмеження відображення – повертає максимум 100 результатів, тому для найкращого користувацького досвіду було використано бібліотеку Vue-select на частині frontend з можливістю пошуку в випадаючому списку.

Для обробки зображень використовується пакет Intervention/Image з драйвером Imagick. Це нативна PHP-розширена бібліотека, яка надає об'єктно-орієнтований інтерфейс для створення, редагування та конвертації зображень. Вона є потужним та широко використовуваним відкритим програмним забезпеченням для обробки зображень з підтримкою понад 200 форматів зображень, включаючи такі популярні, як JPEG, PNG, GIF, BMP, TIFF та багато інших.

Цей пакет дозволяє маніпулювати зображеннями, змінювати їх розмір, додавати водяні знаки та інше. :

```
function saveImage($record, $file, int $width, $height, string
$mediaCollection)
{
    $image = Image::make($file)
        ->resize($width, $height, function ($constraint) {
            $constraint->aspectRatio();
        })
        ->encode('jpg', 80);
```

```
    $filename = uniqid('img_') . '.jpg';

    $tempPath = storage_path('app/public/uploads/' . $filename);
    $image->save($tempPath);

    $media = $record->addMedia($tempPath)
        ->toMediaCollection($mediaCollection);

    return $media->getUrl();
}
```

Для забезпечення ізольованого та стандартизованого середовища розробки і розгортання використовується Docker. Проєкт контейнеризовано з використанням Docker Compose для оркестрації контейнерів. В його конфігурації додатково ще завантажуюься розширення для PHP, такі як Imagick для правильної роботи створення мініатюр зображень.

Міграції в Laravel представляють собою систему контролю версій для бази даних, що дозволяє командам розробників створювати, змінювати та відстежувати зміни схеми бази даних. Це особливо корисно при роботі в команді або при розгортанні застосунку на різних середовищах. В цьому проекті вони розташовані за директорією database/migrations. Кожен файл містить часову мітку, яка визначає порядок виконання міграцій в БД.

Стандартна структура файлу міграції містить два основні методи:

- up() - виконує зміни в базі даних;
- down() - відкочує зміни, що були внесені методом up().

Міграції виконуються за допомогою наступних команд:

```
# Запуск всіх міграцій, які ще не були виконані
php artisan migrate

# Відкочування останньої міграції
php artisan migrate:rollback

# Відкочування всіх міграцій та запуск їх заново
php artisan migrate:fresh
```

Приклад міграції:

```
return new class extends Migration
{

    public function up(): void
    {
        Schema::create('products', function (Blueprint $table) {
            $table->id();
```

```
            $table->string('name');
            $table->string('slug')->unique();
            $table->string('image')->nullable();
            $table->text('description')->nullable();
            $table->decimal('price', 10, 2)->default(0);
            $table->decimal('discountPercentage', 10, 2)->default(0)-
>nullable();
            $table->integer('stock')->default(0);
            $table->unsignedBigInteger('categoryId')->unsigned()->nullable();
            $table->timestamps();

            $table->foreign('categoryId')->references('id')-
>on('categories');
        });
    }


    public function down(): void
    {
        Schema::dropIfExists('products');
    }
};
```

Для подальшого вдосконалення продукту можна додатково реалізувати кращу інтеграцію з «Новою Поштою» для автоматичного створення накладних для посилок, реалізацію відгуків та інтеграцію з платними сервісами.

Реалізовано наступний функціонал:

– створення та перегляд товарів;

– головна сторінка зі всіма товарами;

– кошик із можливістю видалення/додавання товарів та збереженням даних на сервері та у користувача;

– список схожих товарів;

– багатомовність.

3.3 Вимоги до технічних та програмних засобів

Для ефективної розробки та налагодження застосунку використовувались такі програмні та технічні засоби.

– персональний комп'ютер або ноутбук або сервер з процесором не нижче Intel Core i3 (або еквівалент);

– мінімум 8 ГБ оперативної пам'яті;

– накопичувач об'ємом від 100 ГБ для встановлення операційної системи, інструментів та зберігання робочих файлів проєкту.

Програмні засоби:

– операційна система (Windows 10/11, macOS або дистрибутив Linux, наприклад, Ubuntu, тому що запускається через Docker);

– інтегроване середовище розробки (IDE) PHPStorm.

– система керування версіями Git та клієнт для роботи з нею (наприклад, Git Bash, або вбудований клієнт IDE);

– Docker Desktop (або Docker Engine та Docker Compose) для віртуалізації середовища розробки, включаючи базу даних та вебсервер;

– браузери для тестування (наприклад, Google Chrome, Mozilla Firefox).

Для стабільної та надійної роботи вебзастосунку в умовах експлуатації потрібні такі ресурси:

– віртуальний або фізичний сервер (або хмарний екземпляр);

– процесор з мінімум 2 ядрами;

– ОЗУ мінімум 1 ГБ RAM.

– загальний обсяг сховища не менше 20 ГБ;

– стабільне підключення до мережі Інтернет зі швидкістю, достатньою для обслуговування очікуваного трафіку.

Програмні засоби:

– серверна операційна система (наприклад, Ubuntu Server 20.04 LTS або новіша, Debian);

– вебсервер Nginx;

– мова програмування PHP (версія, що відповідає вимогам Laravel 11, при розробці використано PHP 8.2);

– система управління базами даних MySQL Server;

– Composer для управління залежностями PHP.

3.4 Розгортання та встановлення системи

Розгортання та встановлення системи передбачає виконання низки кроків, спрямованих на підготовку програмного середовища до функціонування. Для успішного запуску проекту необхідно дотримуватися наступної послідовності дій.

Клонування репозиторію: Першим кроком є отримання копії програмного коду. Це здійснюється за допомогою системи контролю версій Git шляхом клонування відповідного репозиторію. Команда для виконання цієї операції у терміналі виглядає наступним чином:

```
git clone <URL репозиторію>
cd <назва папки>
```

Встановлення залежностей Composer: Проект використовує менеджер залежностей Composer для управління PHP-бібліотеками. Для встановлення всіх необхідних пакетів слід виконати команду:

```
composer install
```

Ця команда зчитує файл composer.json, який містить перелік залежностей, та завантажує їх до каталогу vendor.

Встановлення залежностей npm: Frontend-частина проекту використовує Node.js та менеджер пакетів npm. Для встановлення JavaScript-залежностей необхідно виконати наступну команду у кореневому каталозі проекту:

```
npm install
```

Ця команда використовує файл package.json для завантаження та встановлення необхідних JavaScript-пакетів до каталогу node_modules.

Побудова Docker-контейнерів: для забезпечення відтворюваного середовища розробки та спрощення процесу розгортання використовується Docker та Docker Compose. Побудова Docker-контейнерів здійснюється за допомогою команди:

```
docker compose build
```

Ця команда створює образи контейнерів на основі файлів Dockerfile, які визначають конфігурацію кожного сервісу (наприклад, вебсервер, база даних).

Виконання міграцій бази даних: проєкт використовує міграції Laravel для керування схемою бази даних. Після запуску контейнера бази даних необхідно застосувати міграції для створення необхідних таблиць. Це можна зробити за допомогою наступної команди, виконаної у контейнері PHP (наприклад, через docker compose exec app php artisan migrate):

```
php artisan migrate
```

Збірка frontend-частини: Frontend-частина (CSS, JavaScript) потребує збірки та міміфікації коду для оптимальної роботи у браузері. Для цього використовується команда npm:

```
npm run build
```

Ця команда запускає процес збірки, налаштований у файлі package.json, і генерує готові до використання файли у каталозі public. Якщо ж потрібно відлагодити код, потрібно запустити команду:

```
npm run dev
```

Створення адміністративного користувача: для доступу до адміністративної панелі необхідно створити обліковий запис адміністратора. Це можна зробити за допомогою спеціальної команди Artisan:

```
php artisan make:filament-user --name=admin
```

Після виконання команди система запропонує ввести email та пароль для нового адміністративного користувача.

Створення символічного посилання для сховища: для забезпечення доступу до файлів, що завантажуються користувачами, з публічного каталогу необхідно створити символічне посилання на директорію storage. Це робиться за допомогою команди:

```
php artisan storage:link
```

Створення директорії для завантажень: для зберігання завантажених файлів користувачів необхідно створити відповідну директорію у сховищі:

```
mkdir storage/app/public/uploads
```

Якщо виникає помилка при завантаженні зображень, потрібно вимкнути перевірк підпису. Для цього слід відкрити файл vendor/livewire/livewire/src/Features/SupportFileUploads/FileUploadController.php та закоментувати рядок 24: abort_unless(request()->hasValidSignature(), 401);.

```
//vendor/livewire/livewire/src/Features/SupportFileUploads/FileUploadControll
er.php
// ...
// abort_unless(request()->hasValidSignature(), 401);
// ...
```

Цей крок є тимчасовим заходом для спрощення розробки та не рекомендується для використання на сервері через потенційні ризики безпеки.

Після успішного завершення всіх етапів встановлення, для запуску застосунку необхідно виконати наступну команду:

```
docker compose up -d
```

Ця команда запускає Docker-контейнери у фоновому режимі. Після цього застосунок буде доступний за відповідною URL-адресою (наприклад, http://localhost:8444, якщо налаштування Docker не були змінені).

Перед запуском переконайтеся, що файл .env у кореневому каталозі проекту містить правильні налаштування, включаючи дані для підключення до бази даних. Для зразка у головній директорії є файл .env.example з усіма полями.

У випадку виникнення будь-яких проблем під час встановлення або запуску, слід звернутися до журналу помилок Docker та Laravel, які можуть містити інформацію про помилки.

3.5 Тестування вебзастосунку

Тестування є критично важливим етапом у життєвому циклі розробки програмного забезпечення, особливо для вебзастосунків, де взаємодія між клієнтською та серверною частинами відбувається переважно через API.

Тестування API дозволяє верифікувати коректність роботи бізнес-логіки, обробки даних та взаємодії компонентів системи на серверному рівні, незалежно від користувацького інтерфейсу.

Для тестування API розробленого вебзастосунку було використано інструмент Postman. Postman є популярною платформою, що надає зручний графічний інтерфейс для створення, відправлення HTTP-запитів, аналізу відповідей сервера та автоматизації тестів. Метою цього етапу було забезпечення функціональної коректності, надійності та відповідності API визначеним вимогам.

Щоб не втратити кінцеві точки API, було створено колекцію з основними операціями взаємодії з сервером через REST API. Основними використаними методами були:

- HTTP GET;
- HTTP POST;
- HTTP PUT;
- HTTP DELETE;
- HTTP PATCH.

На рисунку 3.9 зображений інтерфейс Postman. Зліва, у боковому меню, підпункт 1, зображені існуючі колекції, підпапки до кожної та власне самі методи для тестування. У підпункті 2 розташоване вікно для відображення відповіді від серверу, а у підпункті 3 – власне самий запит: адреса, до якої буде звертатись застосунок,

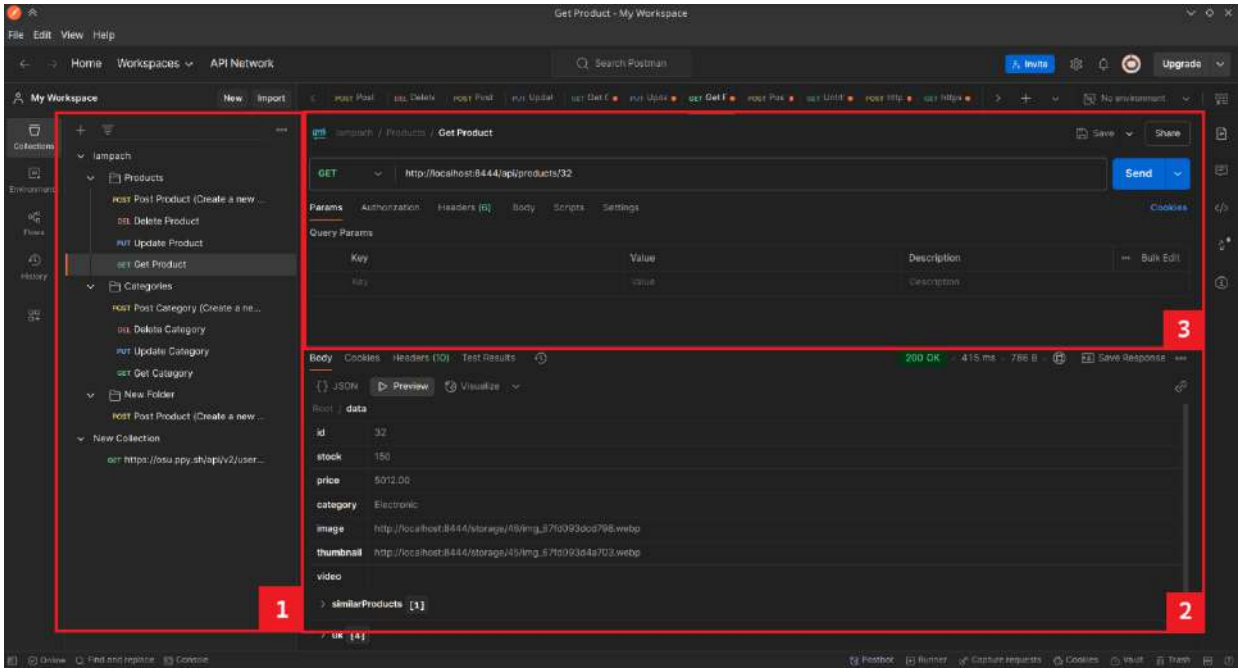


Рисунок 3.9 – Інтерфейс Postman

Було виконано тестування запитів за такими кінцевими точками:

Замовлення (/api/orders):

- GET / - отримання списку замовлень;
- GET /{id} - отримання замовлення за ID;
- POST / - створення нового замовлення.
-

Кошики (/api/carts):

- GET /{id} - отримання кошика за ID;
- POST / - створення/додавання до кошика;
- DELETE /{id} - видалення кошика за ID;
- PATCH /items/{id} - оновлення кількості товару в кошику;
- DELETE /items/{id} - видалення товару з кошика.

Користувачі (/api/users):

- GET / - створення/отримання fingerprint користувача;
- POST /fingerprint - показ даних за відбитком браузера;
- POST / - створення нового користувача;

– PATCH /{id} - оновлення даних користувача.

–

Нова Пошта (/api/nova):

– GET / - отримання списку міст;

– GET /city - отримання відділень у пунктів.

FAQ (/api/faq):

– GET / - отримання списку питань та відповідей.

Протягом цього етапу було виявлено декілька проблем з роботою API «Нової Пошти», одна із них – не повертало правильний список міст при звертанні до кінцевої точки /city, точніше – повертало всі населені пункти, включно з тими, які не мають відділень «Нової Пошти». Проблема була вирішена використанням іншого методу API, який вже перевіряв, чи присутнє хоча б одне відділення.

Використання Filament суттєво спростило процес розробки, так як тут зразу є нявні компоненти для CRUD (Create, Read, Update, Delete) операцій, управління ресурсами, формами, таблицями та іншими елементами інтерфейсу. Через цю особливість додаткових кінцевих точок для продуктів, категорій та іншого не були створені, Filament сам керував цими операціями. Також не потрібно було додатково створювати перевірку автентифікації користувача і чи він може виконувати ту чи іншу операцію.

Тестування спрямоване на перевірку коректності функціонування конкретної імплементації панелі відповідно до вимог проєкту, а не на тестування самого компоненту (оскільки Filament є добре протестованою бібліотекою).

Було протестовано створення кожної сутності бази даних. Виявлено і виправлено такі проблеми:

– не зберігався переклад товару і не відправлявся користувачу на сайт;

– неправильно працювало відображення товарів у замовленні (проблема була у Filament/Repeater та його неправильного використання);

– некоректна валідація даних у формі редагування пов’язаних записів через батьківську форму.

Ці проблеми були вирішенні за допомогою правильно встановлених відносин між батьківськими та пов’язаних з ними моделями та формами, після чого дані тепер коректно записуються у базу даних та правильно повертаються користувачу.

Тестування користувацької частини дипломного проєкту включає перевірку користувацького інтерфейсу та взаємодії з бекендом, щоб переконатися, що все працює коректно і відповідає вимогам. На цьому етапі важливо перевірити, чи всі компоненти працюють коректно, чи дані правильно відображаються, чи правильно він працює на мобільному інтерфейсі та перевірити продуктивність з відображенням на різних браузерах.

Тестування проводилось за допомогою трьох браузерів: Chrome, Firefox, Epiphany (так як він побудований на рушії AppleWebKit і дозволить побачити більшість багів, які можуть виникнути в Safari).

Основними проблемами були:

– деякі компоненти сторінки могли вилізти поза екран (як на екрані комп’ютера, так і на телефоні);

– була відсутність зупинити відео на головній сторінці, так як кнопка для зупинки знаходилась нижче інших компонентів;

– неправильно відображалась головна сторінка на Firefox через відсутність стилю flex.

Основне завдання Telegram-бота – автоматично надсилати повідомлення у визначений чат або користувачу щоразу, коли на сайті оформлюється нове замовлення. Повідомлення містить ключову інформацію про це замовлення.

На початкових етапах тестування було виявлено, що повідомлення про нові замовлення не надсилалися ботом. Причиною виявився невірний ідентифікатор чату або користувача (Chat ID), до якого бот мав відправляти сповіщення. Бот отримував або некоректний, або відсутній ID, через що API Telegram відхиляв запит на надсилання повідомлення. Проблема була вирішена шляхом отримання правильного Chat ID для цільового чату/користувача.

Інформація про замовлення, що надсилається ботом, включає контактний номер телефону клієнта. Були випадки, коли номер телефону зберігався в базі даних у некоректному форматі (наприклад, з відсутніми цифрами, зайвими символами або без коду країни), оскільки на формі замовлення була недостатньо сувора валідація цього поля. Для забезпечення коректності даних, що передаються ботом, було посилено валідацію поля введення номера телефону на формі оформлення замовлення на сайті.

Під час розробки інтерфейсу було проведено тестування полів введення з метою перевірки коректності обробки даних. Тестувалися обмеження на формат, обов’язковість заповнення, довжину та тип введених значень. Перевірялися реакції системи на порожні поля, некоректні символи та перевищення максимальної довжини. Результати тестування підтвердили відповідність полів вимогам до валідації та безпеки.

ВИСНОВКИ

Було успішно розроблено вебзастосунок "Lampach", який є повноцінним інтернет-магазином з продажу ламп. Проект реалізує ключові функції електронної комерції, включаючи зручний каталог товарів, можливість оформлення замовлень та управління користувачами.

Для реалізації проєкту використано сучасний та ефективний стек технологій:

– HTML, CSS (для структури та візуального оформлення інтерфейсу);

– JavaScript, Vue.js (для розробки інтерактивної клієнтської частини);

– PHP, Laravel (для побудови надійного та масштабованого бекенду, API та логіки сервера).

Було реалізовано наступний функціонал:

– розгорнутий каталог ламп з можливістю перегляду всіх доступних позицій, детальною інформацією про окремий товар;

– функціонал додавання товарів до кошика та оформлення замовлення;

– інтеграцію системи сповіщень про нові замовлення, зокрема, реалізовано надсилання повідомлень через Telegram-бота**,** що забезпечує оперативне інформування менеджера магазину про кожне нове замовлення;

– можливість управління асортиментом товарів.

Базуючись на проведених дослідження області, було розроблено коректну структуру бази даних, оптимізовану для потреб інтернет-магазину, та успішно реалізовано програмний продукт.

Проведено ретельне тестування розробленого вебдодатку, включаючи тестування користувацького інтерфейсу, бекенд-логіки та інтеграції з Telegram-ботом. Результати тестування підтвердили надійність, стабільність та коректність роботи всіх компонентів системи.

Вебзастосунок «Lampach» успішно спроєктований, розроблений та готовий до практичного використання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Wix Blog – What is an Online Store. URL: https://www.wix.com/blog/what-is-an-online-store#:~:text=An%20online%20store%2C%20also%20known,purchase%20and%20complete%20transactions%20electronically. (дата звернення: 16.02.2025).

2. Horoshop – Що таке маркетплейс. URL: https://horoshop.ua/ua/blog/what-is-a-marketplace/ (дата звернення: 17.02.2025).

3. Statista – E-commerce Website Visit and Orders by Device. URL: https://www.statista.com/statistics/568684/e-commerce-website-visit-and-orders-by-device/#:~:text=Mobile%20phones%20dominate%20global%20digital,percent%20of%20online%20shopping%20orders. (дата звернення: 18.02.2025).

4. Fingerprint – Browser Fingerprinting Techniques. URL: https://fingerprint.com/blog/browser-fingerprinting-techniques/ (дата звернення: 20.02.2025).

5. AWEBCO – Hero Section. URL: https://www.awebco.com/blog/hero-section/ (дата звернення: 21.02.2025).

6. Investopedia – Business Logic. URL: https://www.investopedia.com/terms/b/businesslogic.asp (дата звернення: 27.02.2025).

7. Docker. URL: https://www.docker.com/ (дата звернення: 15.03.2025).

8. Laravel – The PHP Framework For Web Artisans. URL: https://www.laravel.com/ (дата звернення: 17.03.2025).

9. Vue.js – The Progressive JavaScript Framework. URL: https://vuejs.org/ (дата звернення 19.03.2025)

10. Git – Git Book – What is Git? URL: https://git-scm.com/book/en/v2/Getting-Started-What-is-Git (дата звернення: 24.03.2025).

11. Microsoft Learn – What is an API? URL: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design (дата звернення: 29.03.2025).

12. Tailwind CSS – Utility-first CSS Framework. URL: https://tailwindcss.com/docs/utility-first (дата звернення: 26.03.2025).

13. Postman – API Development Explained. URL: https://www.postman.com/api-platform/ (дата звернення: 25.03.2025).

14. Mozilla – CORS (Cross-Origin Resource Sharing). URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (дата звернення: 04.04.2025).

15. JetBrains – PhpStorm Features Overview. URL: https://www.jetbrains.com/phpstorm/features/ (дата звернення: 08.04.2025).

16. MySQL – What is MySQL? URL: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html (дата звернення: 08.04.2025).

17. Composer – Dependency Manager for PHP. URL: https://getcomposer.org/doc/00-intro.md (дата звернення: 09.04.2025).

18. NGINX – What is NGINX? URL: https://www.nginx.com/resources/glossary/nginx/ (дата звернення: 10.04.2025).

19. intl-tel-input – International Telephone Input Plugin Documentation. URL: https://intl-tel-input.com/ (дата звернення: 15.04.2025).

20. Nova Poshta API – Official Documentation. URL: https://developers.novaposhta.ua/ (дата звернення: 15.04.2025).

21. Nutgram – PHP Telegram Bot Framework Documentation. URL: https://nutgram.dev/docs/ (дата звернення: 16.04.2025).

22. Laravel Breeze – Minimal and Simple Authentication Starter Kit. URL: https://laravel.com/docs/10.x/starter-kits#laravel-breeze (дата звернення: 19.04.2025).

23. MDN Web Docs – HTML: HyperText Markup Language. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення: 20.04.2025).

24. MDN Web Docs – CSS: Cascading Style Sheets. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 20.04.2025).

25. REST API Tutorial – What is REST API? URL: https://restfulapi.net/ (дата звернення: 20.04.2025).

26. Microsoft Docs – Model-View-Controller (MVC) Pattern. URL: https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/getting-started-with-aspnet-mvc3/cs/intro-to-mvc-application (дата звернення: 20.04.2025).

27. GeeksforGeeks – MVCS Architecture in Web Applications. URL: https://www.geeksforgeeks.org/mvcs-architecture-in-web-application/ (дата звернення: 20.04.2025).

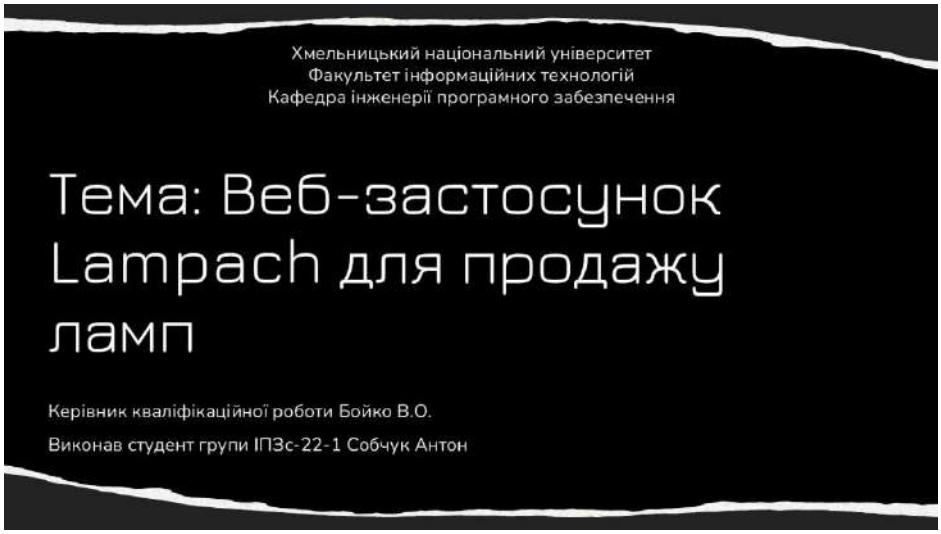
28. MDN Web Docs – Single-page Application (SPA). URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (дата звернення: 21.04.2025).

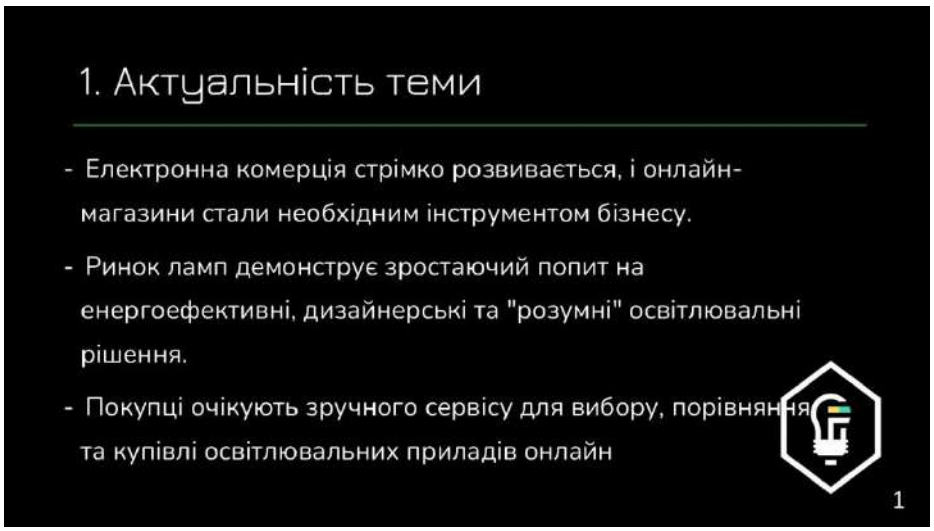
29. MDN Web Docs – JavaScript. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 21.04.2025).

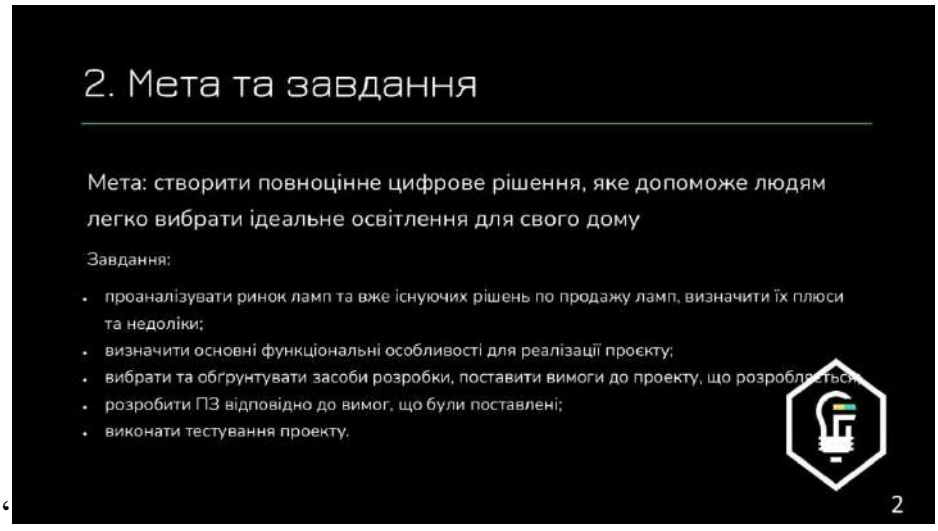
Додаток А

(Обов’язковий)

Презентаційні матеріали







Змістовий аналіз предметної області, її структурних та функціональних особливостей

Серед основних функціональних елементів інтернет-магазинів можна виокремити:

- каталог товарів із категоріями, підкатегоріями, фільтрами та пошуком;
- сторінку товару з фото, описом, характеристиками та наявністю;
- кошик покупця, де зберігаються обрані товари перед оформленням замовлення;
- сторінка з оформленням замовлення з вибором способу доставки та оплати;
- панель адміністратора для керування товарами, замовленнями та користувачами;

3

Аналіз наявного програмно-технічного забезпечення

Критерій	Intellect	Rozetka
Асортимент	Вузькоспеціалізований – лише лампи для дому/офісу	Дуже широкий – товари на будь-який смак і потребу
Технічні характеристики товару	Детальні (цоколь, потужність, температура світла, IP тощо)	Можуть бути обмежені для ламп, часто відсутні вузькі параметри
Інтерфейс	Мінімалістичний, без реклами – нічого не відволікає	Інтуїтивно зрозумілий, але з великою кількістю блоків, які можуть відволікати
Система фільтрів	Відсутня або мінімальна – ускладнює вибір при великому асортименті	Доволі потужна – дозволяє швидко знайти потрібний товар
Оплата	MonoPay, LiqPay, накладений платіж	Різноманітні способи оплати (картки, готівка, розстрочка тощо)

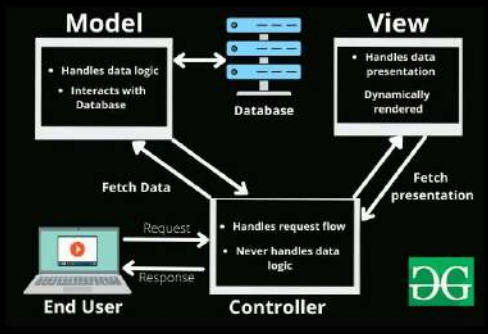
4

Визначення функціональних та нефункціональних вимог до ПЗ

- розробка каталогу товарів з можливістю перегляду детальної інформації про кожний освітлюючий прилад;
- функціонал пошуку, фільтрації та сортування товарів за різними критеріями;
- розробка функціоналу кошика для формування замовлення без авторизації;
- добавлення простої форми для оформлення замовлення;
- адаптивність інтерфейсу для різних типів пристроїв.

5

Вибір типу архітектури



MVC (Model-View-Controller) — це шаблон проєктування, який розділяє додаток на три основні компоненти:

1. **Model (Модель)** — обробляє дані та бізнес-логіку. Взаємодіє з базою даних.
2. **View (Вигляд)** — відповідає за відображення інформації користувачу (інтерфейс).
3. **Controller (Контролер)** — посередник між View та Model. Обробляє запити користувача, викликає відповідну модель та повертає результат у вигляді.

6

Опис декомпозиції, залежностей, інтерфейсів

Компоненти:

- Frontend (Vue.js)
- Backend (Laravel)

Залежності:

- Frontend ↔ Backend через REST API
- Backend ↔3d party API

Інтерфейси:

- REST API (JSON)
- Нова Пошта API
- Telegram API (Nutgram)

7

Проєктування модулів і даних

Контролер	Призначення	Залежить від
CartController	Робота з кошиком: додавання, оновлення, видалення товарів	ProductController
CategoryController	Управління категоріями товарів	ProductController
FAQController	Виведення частих запитань (FAQ) на фронтенд	faq таблиця в БД
NovaPoshta Controller	Вибір міст/відділень доставки через API Нової Пошти	Nova Poshta API
OrderController	Обробка замовлень, надсилання сповіщень у Telegram	OrderProductController, Telegram API
OrderProduct Controller	Пов'язані товари в замовленні, екран з усіма замовленнями	OrderController, ProductController
TelegramController	Конфігурація та інтеграція з Telegram через .env	Telegram API

Основні сутності

- User – інформація про користувача системи;
- Category – категорія товару;
- Product – товар з детальною інформацією;
- Order – замовлення користувача;
- Cart – кошик користувача;
- Media – медіафайли (зображення та відео товарів);
- FAQ – часті запитання.

8

Технології, використані на backend та frontend частині

Технологія	Призначення	Причина вибору
Laravel	Backend-фреймворк	Зручний, безпечний, багато готових рішень
Vue.js	Frontend-фреймворк	Реактивність, швидка розробка, SPA підтримка
Telegram Bot API (Nutgram)	Повідомлення про замовлення	Легкий зв'язок з користувачем, оперативність
Nova Poshta API	Отримання даних для доставки	Автоматизація вибору міста та відділення
MySQL	База даних	Сумісність із Laravel, надійна SQL-система
Vite	Збірка фронтенду	Швидкий старт і гаряче оновлення у Vue
Git + GitHub	Контроль версій	Командна робота, історія змін
Filament	Адмін-панель	Гнучкість та простота в розробці

9

Реалізація БД

Було спроєктовано БД в dbdiagram.io. Структура БД була перенесена за допомогою міграцій Laravel.

10

Вимоги до технічного та програмного забезпечення

- серверна операційна система (наприклад, Ubuntu Server 20.04 LTS або новіша, Debian);
- веб-сервер Nginx;
- мова програмування PHP (версія, що відповідає вимогам Laravel 11, при розробці використано PHP 8.2);
- система управління базами даних MySQL Server;
- Composer для управління залежностями PHP.

11

Тестування ПЗ

1. API (Postman):

- Перевірка коректності роботи REST API.
- Основні методи: GET, POST, PUT, DELETE, PATCH.
- Виявлено та виправлено помилки з кінцевою точкою /api/nova/city.

2. Адмін-панель (Filament):

- Тестування CRUD-операцій та форм.
- Проблеми: не зберігались переклади, неправильна валідація, помилки з Repeater.
- Вирішено через правильну конфігурацію моделей та форм.

3. Користувацький інтерфейс:

- Перевірка в Chrome, Firefox, Safari.
- Виправлено помилки відображення, стилів та відео компонента.

4. Telegram-бот:

- Перевірка надсилання повідомлень про замовлення.
- Виявлено помилку з Chat ID, виправлено.

12

Висновки

У межах проєкту було реалізовано повноцінний функціонал: каталог, сторінка товару з детальним описом; кошик і система оформлення замовлень; адмін панель для керування товарами, замовленнями та користувачами. Система готова до подальшого масштабування та інтеграції з платіжними сервісами.

13

Дякую за увагу!

Додаток Б

Код проєкту

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;

class Cart extends Model
{
    protected $fillable = [
        'userId',
        'status',
    ];

    public function user() : BelongsTo {
        return $this->belongsTo(User::class, 'userId');
    }

    public function cartItems() : HasMany {
        return $this->hasMany(CartItem::class, 'cartId');
    }
}
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class CartItem extends Model
{
    protected $fillable = [
        'cartId',
        'productId',
        'quantity',
        'price'
    ];

    public $timestamps = false;
    const CREATED_AT = 'created_at';

    public function cart() : BelongsTo {
        return $this->belongsTo(Cart::class, foreignKey: 'cartId');
    }

    public function product() : BelongsTo {
        return $this->belongsTo(Product::class, foreignKey: 'productId');
    }
}
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
```

```
use Illuminate\Database\Eloquent\Relations\HasMany;

class Category extends Model
{
    protected $fillable = [
        'name',
        'slug',
        'shopId',
        'description'
    ];
    public function shop(): BelongsTo {
        return $this->belongsTo(Shop::class);
    }

    public function products(): HasMany {
        return $this->hasMany(Product::class);
    }
}
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasOne;

class FAQ extends Model
{
    protected $table = 'faq';
    protected $fillable = [
        'label',
        'description',
    ];

    public function translation():hasOne {
        return $this->hasOne(FAQTranslation::class,'faqId');
    }
}
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class FAQTranslation extends Model
{
    protected $table = 'faq_en';

    protected $fillable = [
        'faqId',
        'label',
        'description'
    ];
}
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;
```

```
class Order extends Model
{
    protected $fillable = [
        'userId',
        'totalAmount',
        'status',
        'firstName',
        'surname',
        'phoneNumber',
        'cityRef',
        'warehouseSettlementRef'
    ];

    public function user(): BelongsTo {
        return $this->belongsTo(User::class, 'userId');
    }

    public function orderProducts(): HasMany {
        return $this->hasMany(OrderProduct::class, 'orderId');
    }
}
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class OrderProduct extends Model
{

    protected $fillable = [
        'orderId',
        'productId',
        'quantity',
        'price'
    ];

    public $timestamps = false;
    const CREATED_AT = 'created_at';

    public function order(): BelongsTo {
        return $this->belongsTo(Order::class, 'orderId');
    }

    public function product(): BelongsTo {
        return $this->belongsTo(Product::class, 'productId');
    }
}
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Intervention\Image\Facades\Image;
use Spatie\MediaLibrary\HasMedia;
use Spatie\MediaLibrary\InteractsWithMedia;
use Spatie\MediaLibrary\MediaCollections\Models\Media;
```

```
class Product extends Model implements HasMedia
{
    use InteractsWithMedia;
    protected $fillable = [
        'name',
        'slug',
        'description',
        'shortDescription',
        'technicalSpecifications',
        'price',
        'discountPercentage',
        'stock',
        'categoryId'
    ];

    public function registerMediaConversions(?Media $media = null): void
    {
        $this->addMediaConversion('thumb')
            ->width(368)
            ->height(232)
            ->sharpen(10);
    }

    public function category(): BelongsTo
    {
        return $this->belongsTo(Category::class, 'categoryId');
    }

    public function cartItems(): HasMany {
        return $this->hasMany(CartItem::class);
    }

    public function orderProducts(): HasMany {
        return $this->hasMany(OrderProduct::class);
    }

    public function similarProducts()
    {
        return $this->belongsToMany(Product::class, 'similar_products',
'productId', 'similarProductId');
    }

    public function similarToProducts()
    {
        return $this->belongsToMany(Product::class, 'similar_products',
'similarProductId', 'productId');
    }

    public function translation() {
        return $this->hasOne(ProductTranslation::class, 'productId');
    }
}
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class ProductTranslation extends Model
{
    protected $table = 'products_en';
```

```
    protected $fillable = [
        'productId',
        'name',
        'description',
        'shortDescription',
        'technicalSpecifications',
    ];
}
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasMany;

class Shop extends Model
{
    protected $fillable = [
        'name'
    ];

    public function categories(): HasMany {
        return $this->hasMany(Category::class);
    }
}
<?php

namespace App\Models;

use Filament\Models\Contracts\FilamentUser;
use Filament\Panel;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class User extends Authenticatable implements FilamentUser
{
    /** @use HasFactory<\Database\Factories\UserFactory> */
    use HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var list<string>
     */
    protected $fillable = [
        'name',
        'password',
        'token',
        'email',
    ];

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var list<string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];
```

```
    /**
     * Get the attributes that should be cast.
     *
     * @return array<string, string>
     */
    protected function casts(): array
    {
        return [
            'email_verified_at' => 'datetime',
            'password' => 'hashed',
        ];
    }

    public function cart() : \Illuminate\Database\Eloquent\Relations\HasOne {
        return $this->hasOne(Cart::class, 'userId');
    }

    public function canAccessPanel(Panel $panel): bool
    {
        return $this->name == 'admin';
        //todo: password verification and .env implementation
    }
}
<?php

namespace App\Http\Controllers;

use App\Http\Requests\StoreCartRequest;
use App\Http\Requests\UpdateCartRequest;
use App\Http\Resources\CartResource;
use App\Http\Services\CartItemsService;
use App\Http\Services\CartService;
use Illuminate\Http\JsonResponse;
use Illuminate\Http\Request;

class CartController extends Controller
{
    protected $cartService;
    private $cartItemsService;

    public function __construct(CartService $cartService, cartItemsService
$cartItemsService) {
        $this->cartService = $cartService;
        $this->cartItemsService = $cartItemsService;
    }

    public function show($id) {
        $cart = $this->cartService->getById($id);
        return response()->json(CartResource::make($cart));
    }

    public function store(StoreCartRequest $request): JsonResponse {
        $request->validated();
        $cart = $this->cartService->store($request->toArray());
        return response()->json(CartResource::make($cart));
    }

    public function update(UpdateCartRequest $request, $id): JsonResponse {
        $request->validated();
        $cart = $this->cartItemsService->update($request->toArray(), $id);
```

```
        return response()->json($cart);
    }

    public function destroy($id) {
        $cart = $this->cartService->destroy($id);
        return response()->json($cart);
    }

    public function remove($cartItemId) {
        $cart = $this->cartItemsService->remove($cartItemId);
        return response()->json($cart);
    }
}
<?php

namespace App\Http\Controllers;

use App\Http\Requests\StoreCategoryRequest;
use App\Http\Requests\UpdateCategoryRequest;
use App\Http\Services\CategoryService;
use Illuminate\Http\JsonResponse;

class CategoryController extends Controller
{
    protected $categoryService;

    public function __construct(CategoryService $categoryService) {
        $this->categoryService = $categoryService;
    }

    public function index(){
        $items = $this->categoryService->getAll();
        return response()->json($items);
    }

    public function show($id): JsonResponse {
        return response()->json($this->categoryService->getById($id));
    }

    public function store(StoreCategoryRequest $request): JsonResponse {
        $request->validated();
        $product = $this->categoryService->store($request->toArray());
        return response()->json($product);
    }

    public function update(UpdateCategoryRequest $request, $id): JsonResponse
{
        $request->validated();
        $product = $this->categoryService->update($id, $request->toArray());
        return response()->json($product);
    }

    public function destroy($id): JsonResponse {
        $product = $this->categoryService->remove($id);
        return response()->json($product);
    }
}
<?php

namespace App\Http\Controllers;
```

```
use App\Http\Resources\FAQResource;
use App\Models\FAQ;
use Illuminate\Http\Request;

class FAQController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    public function index()
    {
        return FAQResource::collection(FAQ::all());
    }
}
<?php

namespace App\Http\Controllers;

use App\Http\Resources\NovaPoshtaWarehouseResource;
use Daaner\NovaPoshta\Models\Address;
use Illuminate\Http\Request;

class NovaPoshtaController extends Controller
{

    protected $adr;
    public function __construct(Address $adr) {
        $this->adr = $adr;
    }

    public function getCities(Request $request) {
        $city = $request->query('city');
        return $this->adr->getCities($city);
    }

    public function getWarehouseSettlements(Request $request) {
        $city = $request->query('cityRef');
        return $this->adr->getWarehouses($city, false);
    }


}
<?php

namespace App\Http\Controllers;

use App\Http\Requests\StoreOrderRequest;
use App\Http\Resources\OrderResource;
use App\Http\Services\OrderService;
use Illuminate\Contracts\Container\BindingResolutionException;
use Illuminate\Http\Request;

class OrderController extends Controller
{
    protected $orderService;

    public function __construct (OrderService $orderService) {
        $this->orderService = $orderService;
    }

    public function index() {
```

```
        return response()->json($this->orderService->getAll());
    }

    /**
     * Store a newly created resource in storage.
     * @throws BindingResolutionException
     */
    public function store(StoreOrderRequest $request) {
        $request->validated();
        $order = $this->orderService->store($request->toArray());

        $telegramController = app()->make(TelegramController::class);
        $telegramController->sendDataToTelegram($request);

        return response()->json(OrderResource::make($order));
    }

    public function show($id)
    {
        $order = $this->orderService->getById($id);
        return response()->json(OrderResource::make($order));
    }

    public function update(Request $request)
    {

    }

}
<?php

namespace App\Http\Controllers;

use App\Http\Requests\storeOrderProductRequest;
use App\Http\Services\OrderProductService;
use App\Http\Services\ProductService;
use Illuminate\Http\Request;

class OrderProductController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    protected $orderProductService;

    public function __construct(OrderProductService $orderProductService) {
        $this->orderProductService = $orderProductService;
    }

    public function index() {
        return response()->json($this->orderProductService->getAll());
    }

    public function show($id) {
        return response()->json($this->orderProductService->getById($id));
    }

    public function store(StoreOrderProductRequest $request) {
        $request->validated();
        $orderProduct = $this->orderProductService->store((array)$request);
        return response()->json($orderProduct);
```

```
    }

    public function update(StoreOrderProductRequest $request, $id) {
        $request->validated();
        $orderProduct = $this->orderProductService->update((array)$request,
$id);
        return response()->json($orderProduct);
    }

}
<?php

namespace App\Http\Controllers;

use App\Http\Requests\StoreProductRequest;
use App\Http\Requests\UpdateProductRequest;
use App\Http\Resources\ProductResource;
use App\Http\Services\ProductService;
use Illuminate\Http\JsonResponse;
use Illuminate\Http\Request;

class ProductController extends Controller
{
    protected $productService;

    public function __construct(ProductService $productService) {
        $this->productService = $productService;
    }

    public function index(Request $request) {
        $perPage = $request->query('per_page', 4);
        $products = $this->productService->paginate($perPage);
        return ProductResource::collection($products);
    }

    public function show($id) {
        $product = $this->productService->getById($id);

        return ProductResource::make($product);
    }

    public function store(StoreProductRequest $request): JsonResponse {
        $request->validated();
        $product = $this->productService->store($request->toArray());
        return response()->json($product);
    }

    public function update(UpdateProductRequest $request, $id): JsonResponse
{
        $request->validated();
        $product = $this->productService->update($id, $request->toArray());
        return response()->json($product);
    }

    public function destroy($id): JsonResponse {
        $product = $this->productService->remove($id);
        return response()->json($product);
    }
}
<?php
```

```
namespace App\Http\Controllers;

use App\Http\Services\ShopService;
use Illuminate\Http\JsonResponse;
use Illuminate\Http\Request;

class ShopController extends Controller
{
    private $shopService;

    public function __construct(ShopService $shopService) {
        $this->shopService = $shopService;
    }

    /**
     * @OA\Get(
     *     path="/api/shops/{id}",
     *     summary="Get shops list",
     *     tags={"Shop"},
     *
     * @OA\Parameter(
     *     name="id",
     *     in="path",
     *     description="ID of the shop to retrieve",
     *     required=true,
     *
     * @OA\Schema(type="integer")
     *   ),
     *
     * @OA\Response(response=200, description="Successful response"),
     * @OA\Response(response=404, description="Shop is not exist")
     * )
     */
    public function show($id) {
        $shop = $this->shopService->getShop($id);
        return response()->json($shop->toJson());
    }

    /**
     * @OA\Post(
     *     path="/api/shops",
     *     summary="Add shop",
     *     tags={"Shop", "Add"},
     *
     * @OA\RequestBody(
     *            required=true,
     *
     * @OA\JsonContent(
     *               type="object",
     *
     * @OA\Property(property="name", type="string", example="Product Name"),
     *           )
     *       ),
     *
     * @OA\Response(response=200, description="Successful added"),
     * @OA\Response(response=400, description="Validation error")
     * )
     */
    public function store(Request $request) {
        try {
            $store = $this->shopService->createShop($request->all());
```

```
            return $store->toJson();
        } catch (\Exception $e) {
            return response()->json(['error' => $e->getMessage()], 400);
        }
    }

    public function update(Request $request, $id) {
        $this->shopService->updateShop($request->all(), $id);
        return $this->shopService->getShop($id);
    }

    public function destroy($id): JsonResponse {
        $result = $this->shopService->deleteShop($id);
        if ($result)
            return response()->json(['message' => 'Shop deleted
successfully'], 200);
        else
            return response()->json(['errors' => 'Shop not deleted'], 400);

    }
}
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use SergiX44\Nutgram\Nutgram;

class TelegramController extends Controller
{
    protected $bot;

    public function __construct(Nutgram $bot)
    {
        $this->bot = $bot;
    }

    public function sendDataToTelegram(Request $request)
    {
        $data = $request->all();
        $message = "🛒 *NEW ORDER RECEIVED* 🛒\n\n";

        // Customer information
        $message .= "👤 *Customer Details:*\n";
        if (isset($data['firstName'])) $message .= "Name:
{$data['firstName']} {$data['surname']}\n";
        if (isset($data['phoneNumber'])) $message .= "Phone:
{$data['phoneNumber']}\n";
        if (isset($data['userId'])) $message .= "User ID:
{$data['userId']}\n";
        if (isset($data['cityRef'])) $message .= "City:
{$data['cityRef']}\n";
        if (isset($data['warehouseSettlementRef'])) $message .= "Delivery:
{$data['warehouseSettlementRef']}\n\n";

        // Order products
        if (isset($data['orderProducts']) &&
is_array($data['orderProducts'])) {
            $message .= "📦 *Products Ordered:*\n";
```

```
            $totalAmount = 0;

            foreach ($data['orderProducts'] as $index => $product) {
                $productNumber = $index + 1;
                $productName = $product['uk']['name'];
                $quantity = $product['quantity'] ?? 1;
                $price = $product['price'] ?? '0.00';
                $itemTotal = $quantity * (float)$price;
                $totalAmount += $itemTotal;

                $message .= "{$productNumber}. {$productName} - {$quantity} x
{$price} = {$itemTotal} UAH\n";
            }

            $message .= "\n💰 *Total Amount: {$totalAmount} UAH*\n";
        }
        $chatId = config('services.telegram.chat_id');
        $this->bot->sendMessage($message, $chatId);
        return response()->json(['success' => true]);
    }
}
<?php

namespace App\Http\Controllers;

use App\Http\Requests\UpdateUserRequest;
use App\Http\Requests\UserRequest;
use App\Http\Resources\UserResource;
use App\Http\Services\UserService;
use Illuminate\Http\Request;

class UserController extends Controller
{
    /**
     * Display a listing of the resource.
     */
    protected $userService;
    public function __construct(UserService $userService) {
        $this->userService = $userService;
    }


    public function showFingerprint(Request $request) {
        $token = $request['token'];
        return response()->json(UserResource::make(
            $this->userService->getByFingerprint($token)
        ));
    }

    public function createAndStoreFingerprint(UserRequest $request) {
        $fingerprint = $request->getBrowserFingerprint();
        return response()->json(UserResource::make($this->userService-
>getByFingerprint($fingerprint)));
    }

    public function store(UserRequest $request) {
        $data = ['token' => $request->getBrowserFingerprint()];
        return response()->json(
            UserResource::make($this->userService->store(($data))));
    }
```

```
    public function update(UpdateUserRequest $request, $id) {
        $user = $this->userService->update($id, $request->validated());
        return response()->json($user->toArray());
    }
}
<?php

namespace App\Http\Services;

use App\Models\CartItem;

class CartItemsService {

    public function update($data, $id) {
        $cartItem = CartItem::findOrFail($id);
        $cartItem->update($data);
        return $cartItem;
    }

    public function remove($id) {
        $cartItem = CartItem::findOrFail($id);
        $cartItem->delete();
        return $cartItem;
    }
}
<?php

namespace App\Http\Services;

use App\Models\Cart;
use App\Models\CartItem;

class CartService {
    public function getById($id) {
        return Cart::findOrFail($id);
    }

    public function store($data) {
        $cart = Cart::firstOrCreate([
            'userId' => $data['userId']
        ]);
        $existCartItem = CartItem::where('productId', $data['productId'])
            ->where('cartId', $cart->id)->first();
        if (!$existCartItem) {
            $createCartItem = [
                'productId' => $data['productId'],
                'quantity' => $data['quantity'],
                'price' => $data['price']];
            $cart->cartItems()->create($createCartItem);
        }
        else {
//            dd( $existCartItem->quantity + $data['quantity'])
            $existCartItem->update(['quantity' => $existCartItem->quantity +
$data['quantity']]);
        }
        return $cart;
    }

    public function destroy($id) {
        return Cart::destroy($id);
```

```
    }

}
<?php

namespace App\Http\Services;

use App\Models\Category;

class CategoryService {

   public function getAll() {
       return Category::all();
   }

   public function getById($id) {
       return Category::findOrFail($id);
   }

    public function getBySlug($slug) {
        return Category::where('slug', '=', $slug)->first();
    }

   public function store($data) {
       try {
           return Category::create($data);
       } catch(\Exception $e) {
           return $e->getMessage();
       }
   }

   public function update($id, $data) {
       try {
           $category = $this->getById($id);
           $category->update($data);
           return $category;
       } catch(\Exception $e) {
           return $e->getMessage();
       }
   }

    /**
     * @throws \Exception
     */
    public function remove($id) {
       $category = $this->getById($id);
       if ($category->delete()) {
           return $category;
       } else
           throw new \Exception("Failed to delete category.");
   }
}
<?php


namespace App\Http\Services;

use App\Models\OrderProduct;
use Exception;

class OrderProductService
```

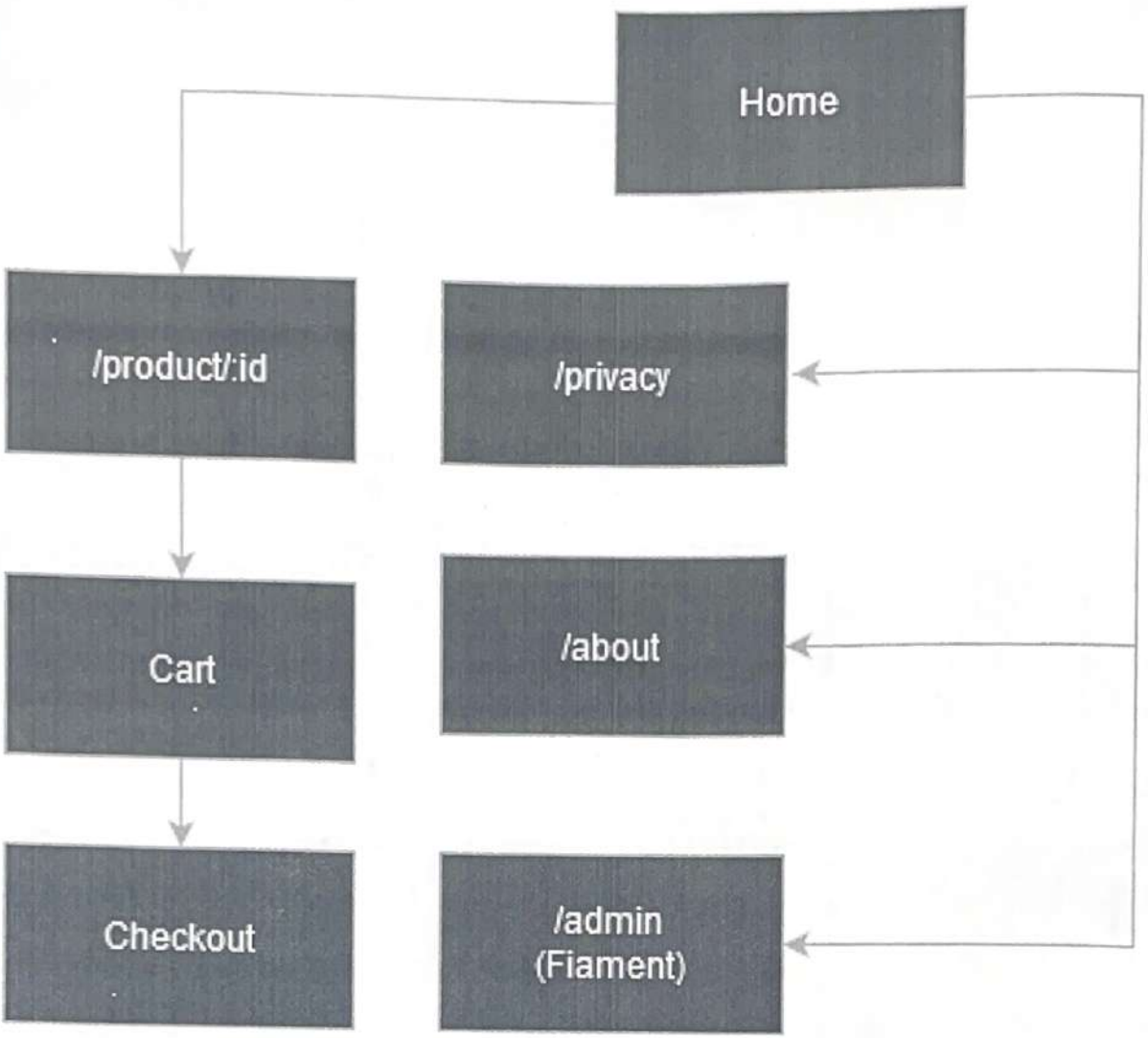
```
{
    public function getAll() {
        return OrderProduct::all();
    }

    public function getById($id) {
        return OrderProduct::findOrFail($id);
    }

    public function store(array $data) {
        try {
            return OrderProduct::create($data);
        } catch (Exception $e) {
            return $e->getMessage();
        }
    }

    public function update($id, array $data) {
        try {
            $orderProduct = $this->getById($id);
            $orderProduct->update($data);
            return $orderProduct;
        } catch (Exception $e) {
            return $e->getMessage();
        }
    }

    /**
     * @throws Exception
     */
    public function remove($id) {
        $orderProduct = $this->getById($id);
        if ($orderProduct->delete()) {
            return $orderProduct;
        } else
            throw new Exception("Failed to delete order product with id
{$id}");
    }
}
<?php

namespace App\Http\Services;

use App\Http\Resources\OrderItemResource;
use App\Models\Order;
use Exception;

class OrderService {
    public function getAll() {
    return Order::all();
}

    public function getById($id) {
    return Order::findOrFail($id);
}
    public function store(array $data) {
        $order = Order::Create([
            'userId' => $data['userId'],
            'status' => 'new',
            'firstName' => $data['firstName'],
            'surname' => $data['surname'],
```

```
            'phoneNumber' => $data['phoneNumber'],
            'cityRef' => $data['cityRef'],
            'warehouseSettlementRef' => $data['warehouseSettlementRef'],
        ]);
        $orderProducts = $data['orderProducts'];
        $order->orderProducts()-
>createMany(OrderItemResource::collection($orderProducts)-
>toArray(request()));
        return $order;
}

    public function update($id, array $data) {
    try {
        $order = $this->getById($id);
        $order->update($data);
        return $order;
    } catch (Exception $e) {
        return $e->getMessage();
    }
}

    /**
     * @throws Exception
     */
    public function remove($id) {
    $order = $this->getById($id);
    if ($order->delete()) {
        return $order;
    }  else
        throw new Exception("Failed to delete order with id {$id}");
}
}
<?php

namespace App\Http\Services;

use App\Models\Product;
use Exception;

class ProductService
{
    public function getAll() {
        return Product::all();
    }
    public function getById($id) {
        return Product::findOrFail($id);
    }
    public function getBySlug($slug) {
        return Product::where('slug', '=', $slug)->first();
    }

    public function store(array $data) {
        try {
            return Product::create($data);
        } catch (Exception $e) {
            return $e->getMessage();
        }
    }

    public function update($id, array $data) {
        try {
```

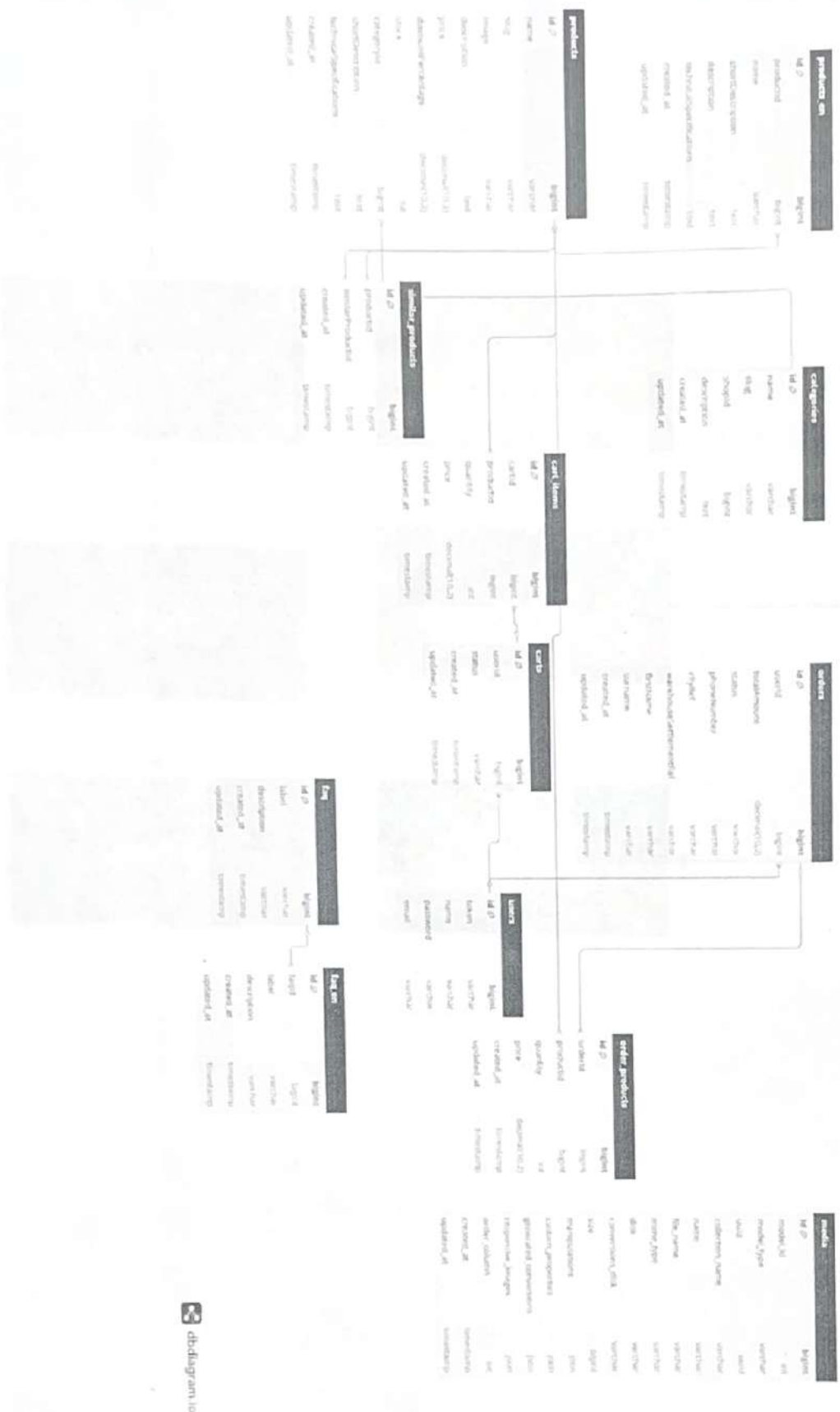
```
            $product = $this->getById($id);
            $product->update($data);
            return $product;
        } catch (Exception $e) {
            return $e->getMessage();
        }
    }
    public function paginate(int $perPage) {
        return Product::with('media')->paginate($perPage);
    }
    /**
     * @throws Exception
     */
    public function remove($id) {
        $product = $this->getById($id);
        if ($product->delete()) {
            return $product;
        }  else
            throw new Exception("Failed to delete product with id {$id}");
    }
}
<?php

namespace App\Http\Services;
use App\Models\Shop;
use Illuminate\Support\Facades\Validator;
use Illuminate\Validation\ValidationException;

class ShopService
{
    public function getShop(int $id): Shop {
        $shop = Shop::findOrFail($id);

        return $shop;
    }

    public function createShop(array $data): Shop {
        $validator = Validator::make($data, [
            'name' => 'required|string|max:255',
        ]);

        if ($validator->fails()) {
            throw new ValidationException($validator);
        }

        $shop = Shop::create($validator->validated());

        return $shop;
    }

    public function updateShop(array $data, $id): Shop
    {
        $validator = Validator::make($data, [
            'name' => 'required|string|max:255',
        ]); // todo: change to ShopRequest

        if ($validator->fails()) {
            throw new ValidationException($validator);
        }

        $shop = $this->getShop($id);
```

```
        $shop->update($validator->validated());

        return $shop;
    }
    public function deleteShop($id): bool {
        try {
            $shop = $this->getShop($id);
            $shop->delete();

            return true;
        } catch (\Exception $exception) {
            return false;
        }
    }
    public function findShopById(int $id): ?Shop {
        return Shop::findOrFail($id);
    }
}
<?php

namespace App\Http\Services;

use App\Models\User;
use Exception;

class UserService {
    public function getAll() {
        return User::all();
    }

    public function getById($id) {
        return User::findOrFail($id);
    }

    public function getByFingerprint($fingerprint) {
        return User::firstOrCreate([
            'token' => $fingerprint]);
    }

    public function store(array $data) {
        return User::firstOrCreate($data);
    }

    public function update($id, array $data) {
        $user = $this->getById($id);
        $user->update($data);
        return $user;
    }

    /**
     * @throws Exception
     */
    public function remove($id) {
        $user = $this->getById($id);
        if ($user->delete()) {
            return $user;
        }  else
            throw new Exception("Failed to delete user with id {$id}");
    }
}
```

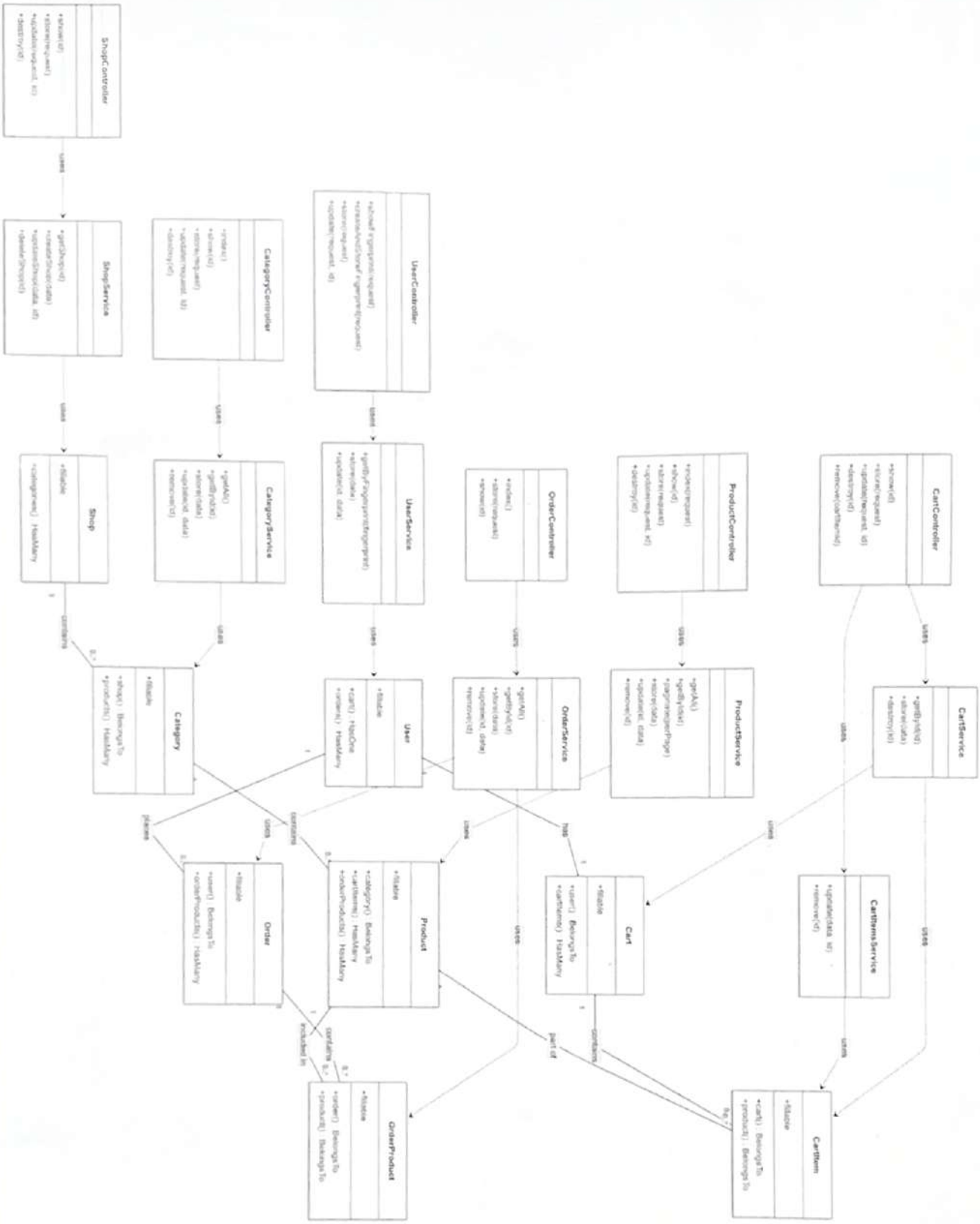
ГРАФІЧНІ МАТЕРІАЛИ



Змін	Аркуш	№ докум.	Підпис	Дата	КППІ.2201125.01.05.Е8		
Розробив		Собчук А.В.		01.06	Вебзастосунок для продажу освітлювальних приладів Схема маршрутизації клієнтської частини вебзастосунку	Літ	Арк 2 / Аркушів 3
Керівник		Бойко В.О.		01.06		Н	
Н. Контр.		Яшина О.М.		01.06		ХНУ.ІПЗс-22-1	
Затвер.		Бедратюк Л.П.		01.06			



Змін.	Аркуш	№ докум.	Підпис	Дата	КПП.2201125.01.05.Е8		
Розробив		Собчук А.В.		01.06	Вебзастосунок для продажу освітлювальних приладів Схема бази даних	Літ	Арк 1 / Аркушів 3
Керівник		Бойко В.О.		01.06		Н	
Н. Контр.		Яшина О.М.		01.06		ХНУ.ІПЗс-22-1	
Затверд.		Бедратюк Л.П.		01.06			



					КПП.2201125.01.05.Е8		
Змін	Аркуш	№ докум.	Підпис	Дата			
Розробив		Собчук А.В.		01.06	Вебзастосунок для продажу освітлювальних приладів Діаграма класів	Літ	Арк	Аркушів
Керівник		Бойко В.О.		01.06		Н	3	3
Н. Контр.		Яшина О.М.		01.06		ХНУ.ІПЗс-22-1		
Затвер.		Бедратюк Л.П.		01.06				

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи ІПЗс-22-1
Собчука А.В.
Прізвище, ініціали

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення»: Вебзастосунок для продажу освітлювальних приладів

(керівник роботи – Бойко В'ячеслав Олександрович)
Прізвище, ім'я, по батькові

02.01.2025
Дата

Підпис студента

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Собчука Антона Вікторовича
факультет ІТ, ІІІ курс, група ІПЗс-22-1

ЗАЯВА

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

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

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

02.05.2025
дата

підпис

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ДЕКЛАРАЦІЯ УЧАСНИКА ОСВІТНЬОГО ПРОЦЕСУ

щодо дотримання академічної доброчесності

Цією декларацією я, Собчук Антон Вікторович,

студент III курсу спеціальності 121 – Інженерія програмного забезпечення, група ІПЗс-22-1

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

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

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

02 січня 2025 р.

Підпис

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 5.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 14%**

ID: 244349 Title: БКР_Розробка веб-додатку інтернет-магазину з продажу освітлювальних приладів Added in a DB: 2025-06-09 Authors: Собчуку Антон Heads: Бойко В'ячеслав Consultants: Opponents:	Document		Sum coincidence on the DB	
	Symbols	Lexemes	Symbols	Lexemes
	59348	963	5400 (9%)	74 (8%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes

Протокол аналізу звіту подібності науковим керівником

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

Автор: Собчук Антон

Співавтор:

Назва: БКР_Розробка веб-додатку інтернет-магазину з продажу освітлювальних приладів

Науковий керівник:

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:9.9%

Коефіцієнт подібності 2:1.5%

Мікропробіли: 0

Заміна букв: 14

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-06-07 13:28:14.0

Після аналізу Звіту подібності констатую наступне:

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

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

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

Обгрунтування:

Дата експерт

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «Бакалавр»

Дипломник Собчук Антон Вікторович

Тема Вебзастосунок для продажу освітлювальних приладів

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень _____; кількість сторінок записки _____

1.Короткий зміст пояснювальної записки та прийнятих рішень У вступі кваліфікаційної роботи окреслено актуальність розвитку онлайн-торгівлі, зокрема сегменту товарів для дому та інтер'єру, з акцентом на освітлювальні прилади– лампи. Вказано, що пандемія COVID-19 прискорила цифровізацію продажів, аукраїнський ринок демонструє значне зростання. Метою роботи є створення онлайнмагазину «Lampach», що забезпечить зручний вибір і покупку ламп із широкимасортиментом і технічною інформацією. Для досягнення мети поставлено завдання: аналіз ринку та існуючих рішень, визначення функціональних особливостей, вибірзасобів розробки, розробка програмного забезпечення та його тестування.

2. Висновок про відповідність роботи поставленому завданню Завдання, сформульовані у вступі, включають аналіз ринку та існуючих рішень, визначення функціональних особливостей, вибір технологій, розробку програмного забезпечення та тестування. я. Таким чином, отримані результати в основному відповідаютьпоставленим завданням.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи Розділ 1 систематизує предметну область інтернет-магазинів з акцентом на продаж ламп, аналізує функціональні елементи та наявне програмно-технічне забезпечення. Методологічний підхід обгрунтований, охоплено аналіз аналогів і визначення вимог. Розділ 2 описує архітектуру, декомпозицію модулів, залежності, інтерфейси, використання MVCS і SPA. Технічний підхід є виваженим. Проте відсутній аналіз альтернатив архітектур, опис ролей користувачів, деталізація логіки взаємодії та візуалізація структури. Розділ 3 реалізує функціональні модулі, інтерфейс, базову логіку роботи з товарами, кошиком і замовленнями, а також тестування.

4. Позитивні сторони роботи Робота демонструє комплексний підхід до аналізу предметної області, інтегруючиопис функціональних елементів інтернет-магазинів із фокусом на сегмент ламп, що єринково релевантним і технологічно доцільним. Вибір архітектури MVCS із SPA тазастосування сучасних технологій Laravel і Vue.js свідчить про технічно досконалий і модульний дизайн системи. Реалізація базових функцій роботи з товарами,

кошиком ізамовленнями підтверджує функціональну повноту продукту. Проведене інтеграційнетестування API та UX/UI тестування демонструють прагнення до забезпечення якості та користувацької зручності. Описані мінімальні вимоги до середовища розгортаннята використані сучасні бібліотеки підкреслюють інженерно обґрунтований підхід.

5. Негативні сторони роботи Робота охоплює лише серверну частину без інтеграції з клієнтським інтерфейсом, що ускладнює повну демонстрацію функціоналу. Також було б доцільно реалізувати логування дій користувачів та більше автоматизованих тестів.

6. Оцінка графічного оформлення та пояснювальної записки Графічний матеріал представлений у вигляді діаграм архітектури та моделей бази даних. Пояснювальна записка оформлена відповідно до діючих стандартів, має логічну структуру та достатню глибину розкриття теми.

7. Відгук про кваліфікаційну роботу в цілому Тема роботи є актуальною та має практичну значущість у контексті зростання онлайнторгівлі та сегменту домашнього освітлення. Зміст роботи в цілому відповідає темі і поставленим завданням. Об'єктивність і обґрунтованість представлені на середньому рівні через відсутність повного аналізута чітких висновків. Загалом, робота демонструє компетентність здобувача.

8. Інші зауваження Програмний продукт є функціональним, але обмеженим, з базовою реалізацією,без інтеграції з платіжними системами

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

РЕЦЕНЗЕНТ (ПІБ, посада, місце роботи) Лисенко С.М. доктор технічних наук, професор, заслужене декана факультету інформаційних технологій з науковї та мінсторазної роботи

"09" червня 2025 р. _____
(підпис)