МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ІНТЕРНЕТ-МАГАЗИНУ ПРОДАЖ ШКІРЯНИХ ВИРОБІВ З ВИКОРИСТАННЯМ LARAVEL ТА NUXT.JS»**

Виконав: студент 2 курсу, групи 8.1211-1іпз
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Н.Г. Новиков
(ініціали та прізвище)
Керівник професор кафедри програмної інженерії, професор, д.т.н. Чопоров С.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

« ____ » _____ 2022 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Новикову Нікіті Геннадійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Розробка інтернет-магазину продаж шкіряних виробів з використанням Laravel та Nuxt.js

керівник роботи (проекту) Чопоров Сергій Вікторович, д.т.н., професор
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 04 » травня 2022 року № 500-с

2. Строк подання студентом роботи 30.11.2022

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі, аналіз предметної області.
2. Аналіз вимог до інформаційної системи.
3. Реалізація інформаційної системи.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 04.05.2022

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	20.09.2022	
2.	Збір вихідних даних.	22.09.2022	
3.	Обробка методичних та теоретичних	26.09.2022	
	джерел.		
4.	Розробка першого розділу.	05.10.2022	
5.	Розробка другого розділу.	15.10.2022	
6.	Розробка третього розділу.	10.11.2022	
7.	Оформлення та нормоконтроль	01.12.2022	
	кваліфікаційної роботи.		
8.	Захист кваліфікаційної роботи.	16.12.2022	

Студент _____ Н.Г. Новиков
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.В. Чопоров
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка інтернет-магазину продаж шкіряних виробів з використанням Laravel та Nuxt.js»: 67 с., 40 рис., 1 табл., 11 джерел, 6 додатків.

ВЕБ РОЗРОБКА, ІНТЕРНЕТ ЗАМОВЛЕННЯ, ІНТЕРНЕТ МАГАЗИН, ОФОРМЛЕННЯ ЗАМОВЛЕННЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ШКІРЯНІ ВИРОБИ.

Об’єкт дослідження – інтернет магазин з продажу шкіряних виробів.

Мета роботи: розробка магазину шкіряних виробів з використанням Laravel та Nuxt.js в якості фронтенд фреймворка.

Методи дослідження – об'єктно орієнтований аналіз, методи програмної інженерії, методи аналізу даних, методи проектування, конструювання програмного забезпечення.

У кваліфікаційній роботі розглянуто процес розробки WEB додатка. Проведено аналіз предметної області, технологій та інструментів для вирішення поставленого завдання, розроблені наступні діаграми: прецедентів, діяльності, компоненти, класів та розгортання, компоненти, класів та розгортання, спроектована структура додатка, а також база даних. За результатом виконання кваліфікаційної роботи, реалізований інтернет магазин з продажу шкіряних виробів з використанням фреймворку Laravel, а також Nuxt.js, в якості фронтенд фреймворка. Навички, отримані під час розробки WEB додатка, можуть бути застосовані на реальних проєктах.

SUMMARY

Master's Qualifying Paper «Development of the Online Store using Laravel and Nuxt.js»: 67 pages, 40 figures, 1 table, 11 references, 6 supplements.

WEB DEVELOPMENT, INTERNET ORDER, INTERNET SHOP, CHECKOUT, SOFTWARE, LEATHER PRODUCTS.

The object of the study is an online store selling leather goods.

The aim of the study is to develop a store of leather goods using Laravel and Nuxt.js as a frontend framework.

Research methods – object-oriented analysis, methods of software engineering, data analysis methods, design methods, software design.

In the qualification work, the process of developing a WEB application is considered. The analysis of the subject area, technologies and tools for solving the problem was carried out, developed the following diagrams: precedents, activities, components, classes and deployment, the structure of the application, as well as the database, were designed. As a result of the qualification work, an online store for the sale of leather goods was implemented using the Laravel framework, as well as Nuxt.js, as a frontend framework. The skills gained during the development of a WEB application can be applied to real projects.

ЗМІСТ

ВСТУП

З розвитком комп'ютерних технологій Інтернет міцно зайняв значне місце в нашому житті. У мережі є можливість зробити будь-яку покупку, починаючи від великого придбання і закінчуючи різною дрібницею. Покупки в Інтернеті стали для нас уже чимось звичайним, хоча здавалося на зорі розвитку всесвітньої павутини ніхто і подумати що online покупки стануть такими популярними.

Конкуренція у світі електронної торгівлі примушує сучасні онлайн-магазини прикладати максимум зусиль, щоб втриматися на плаву. Тому інтернет магазин повинен мати всі важливі характеристики, щоб зацікавити потенційних клієнтів.

Серед характеристик сучасних інтернет-магазинів, варто виділити наступне:

– зручний та продуманий користувацький інтерфейс: незручне розташування розділів, складна система фільтрів та сортування, необхідність заповнення різноманітних форм – все це може звести нанівець успіх будь-якого проєкту;

– наявність зворотного зв’язку: кожен покупець бажає бути почутим у випадку виникнення питань чи непорозумінь;

– швидкість та легкість здійснення замовлень: оптимальний варіант – можливість реєстрації на сайті з метою спрощення процесу оформлення покупок;

– інтернет-магазин має бути надійним: підтвердженням надійності віртуального магазину може стати розділ «Відгуки», в якому замовники можуть залишати свої коментарі щодо придбаної продукції.

Мета роботи – аналіз методів програмної інженерії для проектування та розробки інтернет магазину на прикладі реалізації магазину.

В рамках досягнення поставленої мети і обраного варіанту завдання необхідно спроектувати і розробити систему з продажу шкіряних виробів онлайн.

В ході виконання кваліфікаційної роботи повинні набути практичних навичок з розробки конкретної предметної області та практичне застосування реляційних моделей баз даних, а також продемонструвати свої навички роботи з веб технологіями і їх використанням.

Робота складається з трьох розділів.

У першому розділі роботи розглядається технічне завдання (ТЗ), опис того що потрібно розробити.

У другому розділі роботи розглянуто процес проектування проекту. Сюди належить проектування бази даних (сутності, зв'язки), розробка наступних діаграм: прецедентів, діяльності, компонентів, класів та розгортання.

У третьому розділі відведено увагу самого процесу розробки, також торкнулися аспектів тестування програмного забезпечення, його контейнеризація. Наведені приклади певних частин коду, пояснення того, як що працює.

1 АНАЛІЗ ВИМОГ

1.1 Аналіз предметної області

Адміністратор інтернет-магазину є головною ланкою в бізнес-процесі, який має підтримувати актуальну базу даних про наявність товару для більшого попиту клієнтів. Адміністратор працює з великим обсягом інформації, веде базу клієнтів, приймає заявки клієнтів, інформує працівників складу про замовлення, передає інформацію про замовлення кур'єру.

Кожен факт про замовлення виробу, фіксується в системі, таблиці замовлень. Кожне замовлення прикріплюється до того користувачу, який зробив покупку, що дозволить йому, зайшовши в свій аккаунт ознайомитися з історією замовлень. Після оформлення замовлення, з користувачем зв'яжеться оператор для подальшого оформлення замовлення, надалі функціонал буде розширено, а саме додана можливість здійснювати покупку прямо на сайті за допомогою платіжної системи.

Під час етапу дослідження варто звертати увагу на такі пункти:

1) вибір ніші: ніша – це товари, які онлайн магазин буде продавати;

2) цільова аудиторія: перед розробкою, варто ознайомитися з потенційними клієнтами сайту;

3) вибір постачальників: пошук постачальника – один з найважливіших етапів, адже надійність, чесність, оперативність обраного партнера може надавати прямий вплив на прибутковість онлайн магазину;

4) асортимент товарів: пошуку постачальників передує складання списку товарів для онлайн-магазину, потрібно скласти приблизний каталог речей;

5) організаційні моменти: такі як оформлення замовлення, зв'язок із покупцем.

1.2 Аналіз вимог до програмного забезпечення

До матеріалів для виробів зі шкіри висуваються такі вимоги:

– функціональні;
– гігієнічні;
– естетичні;
– економічні;
– вимога надійності.

Зупинимося детальніше на кожній з цих вимог.

Функціональні вимоги. Характеризують придатність матеріалу для виконання основної функції виробу. Функціональні властивості матеріалів для виробів зі шкіри можуть бути представлені комплексом фізичних (у тому числі гігієнічних) та механічних властивостей. Фізичними називають такі властивості матеріалу, які характеризують його структуру та ставлення до недеформуючих впливів (наприклад, щільність, пористість, паропроникність та ін.). Механічними називають такі властивості матеріалу, які показують відношення до дії різних прикладених до них механічних зусиль, що викликають його деформацію або руйнування (наприклад, межа міцності при розтягуванні та подовження при розриві, твердість, стирання та ін.).

Гігієнічні вимоги. Основною вимогою до матеріалів для виробів, які повинні мати певні гігієнічні властивості, є наявність пористої структури. Пористі матеріали мають меншу масу та теплопровідність, ніж непористі, на їх виробництво витрачається менше сировини. Матеріали, що мають розгалужену пористу структуру із взаємозалежними порами (наприклад, шкіра), паро-, повітро- та вологопроникні. Матеріали з незв'язаними порами (наприклад, гума пориста) не намокають, мають хороші теплоізоляційні властивості, але паронепроникні. Матеріали для виробів зі шкіри по відношенню до дії вологи діляться на гідрофільні та гідрофобні, тобто змочуються і незмочуються водою. Гідрофільні матеріали мають високі показники вологопоглинання та гігроскопічності, а гідрофобні – показник

водостійкості. У зв'язку з цим гідрофільні матеріали (наприклад, шкіру) застосовують для деталей верху, підкладки та устілок, які повинні поглинати потовиділення стопи, а гідрофобні матеріали – для підошв та підборів, які не повинні намокати.

Естетичні вимоги. Включають колір, фактуру поверхні та інші показники зовнішнього вигляду матеріалів, відповідність їх модним напрямкам.

Економічні вимоги. До всіх матеріалів для виробів зі шкіри висувають економічні вимоги, які полягають у можливості максимального безвідходного використання та мінімальної вартості.

Вимога надійності. Деякі механічні властивості матеріалів визначають надійність виробу (наприклад, опору стирання та багаторазовому вигину, стійкість до дії вологи, тепла та інших зовнішніх факторів).

Основні вимоги до якості шкіряної галантереї:

– вироби шкіряної галантереї повинні бути красивими, легкими, міцними, зручними по конструкції, з правильно прикріпленою фурнітурою, що добре діє, виготовлені з доброякісного матеріалу, що має рівномірне і стійке до зовнішніх впливів забарвлення;

– металева фурнітура повинна мати суцільне та міцне покриття, без подряпин, плям, вм'ятин та іржі;

– з'єднання деталей виробів має бути акуратним, без перекосів, рядок – рівний, добре стягнутий, без перепусток, петель і просічок матеріалу.

Основні функції програмного забезпечення, які мають бути реалізовані:

– реалізація каталогу товарів з можливістю фільтрації, зручної навігації;

– у каталозі товарів для кожної групи товарів повинен працювати індивідуальний фільтр. Параметри фільтрації будуть узгоджені кожної групи окремо;

– подання товарів з їх характеристиками – повна інформація про товар на детальній сторінці з можливістю купити;

– реалізація процесу замовлення;

– розширення функціональності буде здійснюватися на вимогу замовника після запуску першого етапу проекту як одиничних доробок чи цілими етапами.

Вимоги до дизайну програмного забезпечення:

– фокус на товарах;

– адаптивний дизайн;

– візуалізація елементів;

– єдиний фірмовий стиль;

– продумане юзабіліті.

1.3 Аналіз технологічного стека, порівняння, характеристика

Стек технологій на яких буде побудований проект виглядає так:

– MySQL 5.7 в якості бази даних [1];

– Laravel 8.5 в якості фреймворка для розробки backend частини сайту [2];

– Nuxt.js 2 як фреймворка для роботи з frontend частиною [3].

Фреймворк Laravel був обран для створення backend частини, оскільки, на мою думку, якщо необхідно створити сайт з нуля, то дане рішення заощадить багато часу, Laravel дозволяє швидко і просто реалізувати додаток будь-якої складності. Розглянемо особливості даного фреймворка.

Особливості Laravel:

1) Template Engine: фреймворк Laravel широко відомий своїми вбудованими полегшеними шаблонами, які використовуються для створення приголомшливих макетів шляхом заповнення інтерактивного контенту;

2) пропонує стабільні структури з декількома віджетами, інтегруючими код CSS і JS. Шаблони Laravel розроблені для створення простих, але складних макетів з виділеними сегментами;

3) підтримка архітектури MVC: Laravel просуває шаблон архітектури MVC, який дозволяє розділити бізнес-логіку і рівні уявлення;

4) шаблон MVC Laravel має кілька вбудованих функцій, підвищує продуктивність додатків і покращує як захист, так і зручність використання;

5) фреймворк Laravel забезпечує дуже надійний захист веб-додатків, він використовує схему хешування паролів, тому пароль ніколи не може бути збережений в базі даних у вигляді простого тексту;

6) використовує «алгоритм хешування Bcrypt» для створення зашифрованого пароля. Крім того, це середовище веб-розробки для PHP використовує структуровані оператори SQL, що запобігають атаки за допомогою SQL-ін'єкції;

7) система міграції бази даних: система міграції Laravel спрямована на розширення структури бази даних веб-додатки без необхідності її повторного створення щораз, коли відбувається зміна коду. Завдяки такій функції ризик втрати даних вкрай мінімальний. Це не тільки дає можливість змінювати структуру бази даних, але також дозволяє використовувати код PHP, а не SQL. Конструктор схем Laravel допомагає створювати таблиці для бази даних і швидко приєднує індекси або стовпці.

Також хотілося б виділити переваги і недоліки Laravel фреймворка.

Переваги:

– кращий варіант фреймворка для розробників PHP;

– швидка і проста для розуміння коротка документація, що робить PHP і Laravel кращим вибором;

– включає в себе функцію впровадження залежностей, що дозволяє проводити швидке тестування і автоматизацію;

– постійно розвивається фреймворк і співтовариство;

– популярність і ком'юніті: цей Framework популярний на всіх ринках, він підтримується величезним співтовариством.

Недоліки:

– Laravel напрочуд повільний;

– відсутні інтегровані шаблони інтерфейсів.

Думаю, буде доречним порівняти Laravel з Node.js.

Node.js – це кроссплатформенная виконавча середа з відкритим вихідним кодом, призначена для створення серверних додатків. Він має вбудовані програми JavaScript, які можна запускати в OS X, Microsoft Windows і Linux в середовищі виконання Node.js.

Особливості Node.js:

1) *асинхронний і керований подіями*: всі API інтерфейси бібліотеки Node.js є асинхронними, тобто неблокуючими. По суті, це має на увазі, що сервер, заснований на Node.js, ніколи не чекає, поки API поверне дані;

2) *однопотоковий і масштабований*: Node.js використовує однопоточну модель циклічних подій. Механізм подій дозволяє серверу реагувати неблокуючим чином і забезпечує високу масштабованість сервера, на відміну від звичайних серверів, які генерують обмежені потоки для обробки запитів;

3) *кросплатформеність*: Node.js є кросплатформним і також може бути перетворений в виконавчий файл, що включає всі його власні залежності, при розробці з правильною структурою;

4) *багата екосистема бібліотек та інструментів:* одна ініціалізація – npm, стандартний менеджер пакетів Node.js – також служить ринком для інструментів JavaScript з відкритим кодом, що відіграє важливу роль у розвитку цієї технології.

У таблиці 1.1 можна побачити порівняння двох інструментів для розробки бекенд частини.

Таблиця 1.1 – Порівняння Laravel та Node js

Laravel	**Node.js**
Це фреймворк PHP MVC	Це середовище виконання JavaScript, заснована на движку виконання JavaScript Google Chrome

Продовження таблиці 1.1

Laravel	**Node.js**
Він трохи відстає в сегментах ринку в порівнянні з Node JS	Це лідер ринку технологій в порівнянні з Laravel
У нього немає моделі введення-виведення, оскільки це просто фреймворк PHP	Використовується неблокірующіх модель введення-виведення, керована подіями, що робить Node js більш ефективним і простим
Ідеально підходить для додатків MySQL і Maria DB	Легкий у роботі з Express JS і ідеально підходить для MongoDB / MongooseJS
Якщо потрібна повномасштабна система для роботи з великим веб-сайтом, заснованим на CMS, вибирайте Laravel	Якщо потрібна компактна архітектура на основі сервісів, використовуйте Node JS
Платформа Laravel з коробки містить Eloquent ORM. Яка дозволяє розробникам веб-додатків виконувати запити до бази даних з синтаксисом PHP замість написання коду SQL	У Node js для цього ви повинні самі вибрати яку ORM ви б вважали за краще використовувати

Після порівняння, можна зробити наступний висновок, що обидва рішення маю як свої плюси, так і мінуси. Все залежить від типу продукту, який потрібно створити і його рівня складності / масштабованості. Варто відзначити, що фреймворк Laravel дозволяє швидше закласти структуру проекту.

Для розробки front-end частини був обраний Nuxt.js, так як він побудований на базі Vue, і дозволяє спростити розробку універсальних і односторінкових сервісів [4]. Розглянемо його ближче.

Переваги:

– *просте створення універсальних програм:* одне з головних достоїнств полягає в тому, що фреймворк полегшує створення універсальних програм;

– *статичний рендеринг:* найбільше нововведення приходить з командою nuxt generate. Вона повністю генерує статичну версію вашого сайту. Фреймворк створить HTML для кожного з ваших Рауса і помістить його в свій власний файл;

– *автоматичне розбиття коду:* фреймворк може генерувати статичну версію вашого сайту зі спеціальною конфігурацією Webpack. Для кожного статично генерується роута (сторінки) він також отримує свій власний файл JavaScript, що містить тільки код, необхідний для запуску;

– *відмінна структура проекту за замовчуванням:* у багатьох невеликих додатках Vue ви керуєте структурою коду, в кращому випадку, в декількох файлах. Структура Nuxt.js за замовчуванням дає вам відмінний старт для організації вашого сервісу в зрозумілій формі;

– *є підтримка SSR:* в режимі SSR, також званому "універсальний" або "ізоморфний", сервер Node.js буде рендерить HTML перед відправкою на клієнт на основі ваших Vue компонентів, замість відтворення на чистому javascript. Використання режиму SSR покращує роботу з SEO, UX і дає безліч інших можливостей (в порівнянні з традиційним SPA клієнтом на Vue).

Недоліки:

– погано задокументований (має мале співтовариство розробників);

– інтеграція призначених для користувача бібліотек з Nuxt.js досить складна;

– відсутність деяких поширених компонентів, такі як карти Google, календар, векторні карти. Деякі компоненти для цього існують, але, як правило, вони не дуже добре підтримуються;

– ви можете запитувати і управляти DOM тільки в певних хуках.

З моєї точки зору дані фреймворк варто порівняти з Next.js.

Next.js – це фреймворк React, який дозволяє створювати рендеринг на стороні сервера і статичні веб-додатки за допомогою React, розглянемо його ближче.

Переваги:

– не завантажує невикористаний код;

– проста маршрутизація на стороні клієнта (на основі сторінок);

– може бути інтегрований з Express або з будь-яким іншим HTTP-сервером Node.js;

– можливість налаштувати власні конфігурації Babel і Webpack;

– вбудована обробка пошукової оптимізації (SEO) сторінок.

Недоліки:

– Next.js не є серверної частиною; якщо вам потрібна внутрішня логіка, така як база даних або сервер облікових записів, ви повинні зберегти це в окремому серверному додатку;

– Next.js – це потужний інструмент, але якщо ви створюєте просте додаток, його використання може виявитися зайвим;

– всі дані повинні бути завантажені як з клієнта, так і з сервера;

– перенесення серверного додатка на Next.js – це не швидкий процес, і в залежності від вашого проекту може знадобитися занадто багато роботи.

Обидва фреймворка надають потужний набір інструментів і готових рішень при створенні UI частини, у кожного є свої сильні і слабкі сторони. Мій же вибір це Nuxt.js.

В якості системи управління базою даних, була обрана MySQL, а не MongoDB, оскільки на мою думку, бази даних SQL краще підходять для багаторядкових транзакцій, а NoSQL краще підходить для неструктурованих даних, таких як документи або JSON. Чому для проекту не була обрана система управління PostgreSQL?

Обрано MySQL оскільки робота з данною СУБД простіше і швидше, але давайте все ж таки розглянемо відмінності MySQL від PostgreSQL:

1) MySQL має підтримку не всіх функцій і можливостей SQL. Це зроблено для того, щоб працювати з MySQL було просто і зручно. PostgreSQL підтримує всі нові стандарти SQL;

2) в MySQL для зберігання даних в таблицях використовуються різні движки. Движок не має впливу на синтаксис запитів і їх виконання. Є підтримка MyISAM, InnoDB, MEMORY, Berkeley DB. PostgreSQL працює тільки на движку storage engine. Таблиці організовані у вигляді об'єктів, а дії виконуються за допомогою об'єктивно орієнтованих функцій;

3) MySQL і Postgresql мають схожий набір, який, звичайно ж, має свої відмінності. У Postgresql типи різноманітніші.

У плані продуктивності варто відзначити MySQL, так як дана СУБД завжди була орієнтована на велику продуктивність, в той час як Postgresql був націлений на велику кількість налаштувань і стандартів.

Підсумувавши всі, можна зробити наступний висновок: використання в проекті наступних технологій: Laravel, Nuxt.js, MySQL – це відмінне поєднання, з даними технологіями можна швидко і без будь-яких зусиль розробити проект будь-якого рівня складності, і взагалі це дуже потужна зв'язка технологій разом.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

У цьому розділі розглядається структура бази даних, її сутності та зв'язки між ними, а також наведені наступні діаграми: прецедентів, діяльності, компонентів, класів та розгортання.

2.1 Структура бази даних

Для обраної предметної області була обрана структура даних, яка графічно представлена на рисунках 2.1 – 2.3 [1, 10].

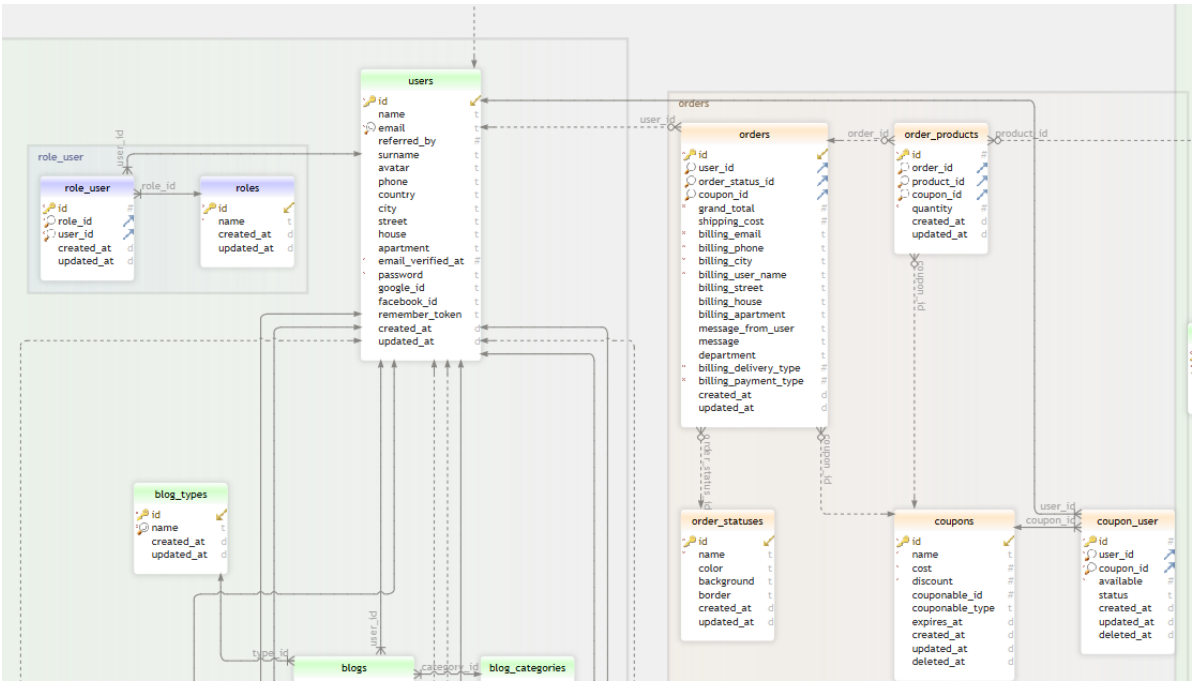

Рисунок 2.1 – Частина сутностей бази даних (users, coupons, roles, тощо)

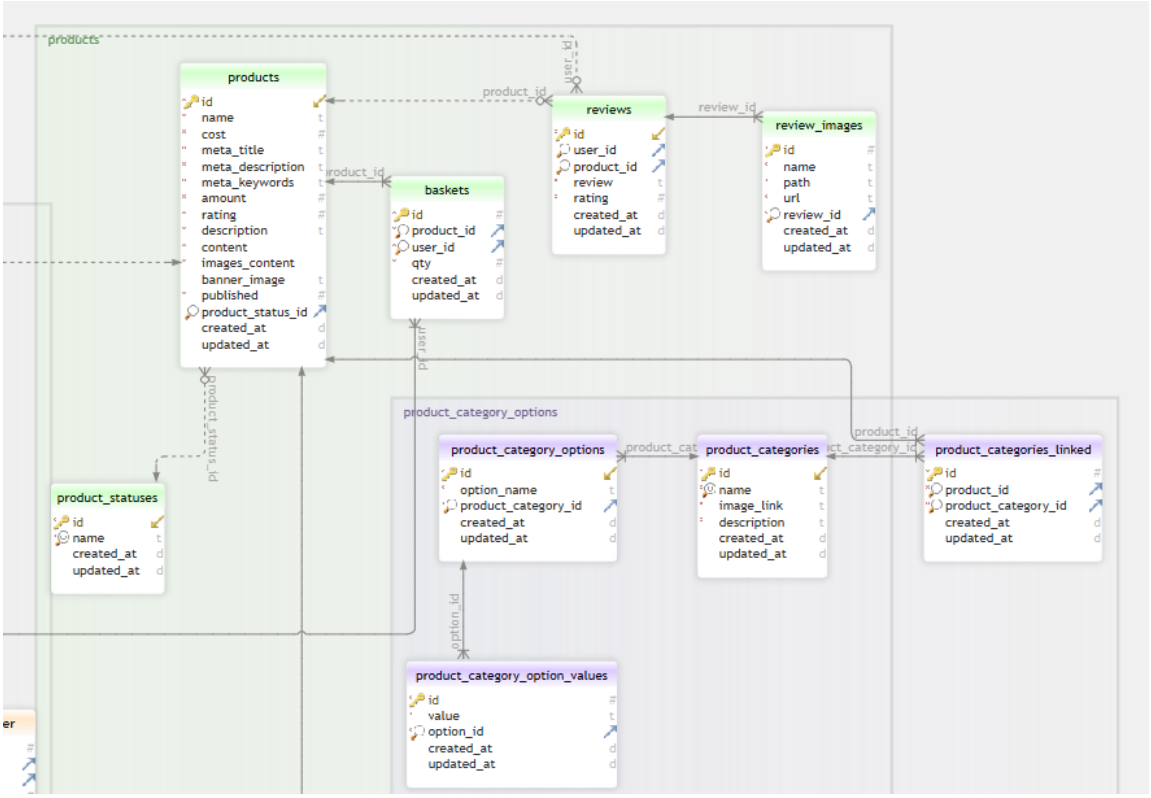

Рисунок 2.2 – Частина сутностей бази даних (products, baskets, product_categories, тощо)

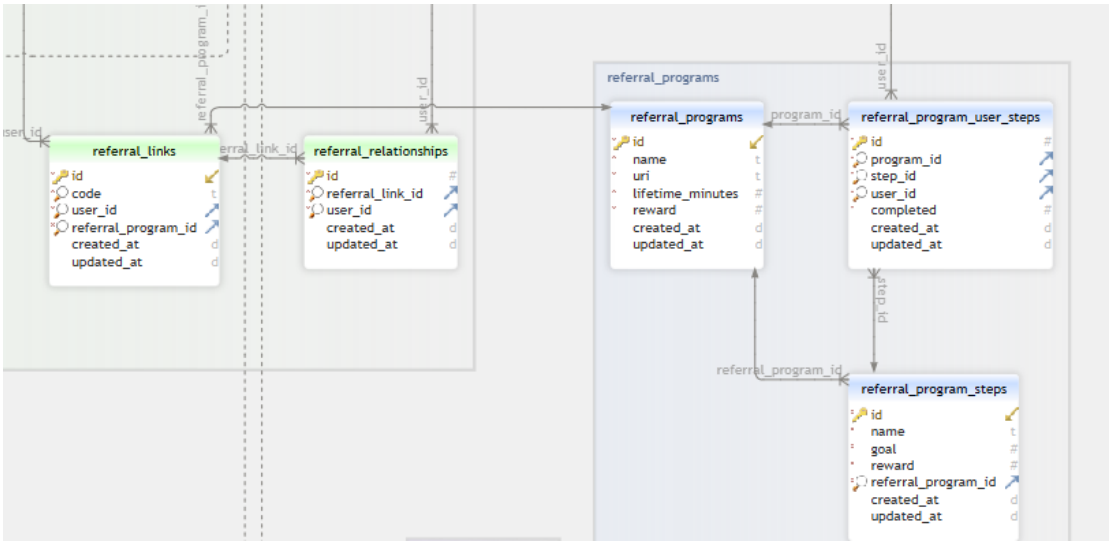

Рисунок 2.3 – Частина сутностей бази даних (referral_programs, referral_links)

Базові сутності:

– users (таблиця користувачів);
– roles (таблиця ролей);
– role_user (таблиця ролей і користувачів);

– products (таблиця продуктів);

– product_statuses (таблиця статусів продуктів);

– product_categories (таблиця категорій продуктів);

– product_categories_linked (сполучна таблиця для таблиць “products” та “categories”);

– orders (таблиця оформлених замовлень);

– order_statuses (таблиця статусів замовлення);

– order_products (таблиця що пов'язує замовлення та продукти);

– referral_links (таблиця для зберігання реферальних посилань);

– referral_programs (таблиця для зберігання реферальних програм);

– referral_program_steps (таблиця для зберігання кроків реферальних програм);

– referral_program_user_steps (таблиця для зберігання кроків користувача у реферальній програми);

– coupons (таблиця для зберігання дисконтів);

– coupon_user (таблиця для асоціації купона з обліковим записом користувача).

Кожна з вищезгаданих сутностей містить низку атрибутів. Зупинимося детальніше на кожній з них.

Users:

– name – ім'я користувача;

– email – email адрес користувача;

– referred_by – ID користувача, завдяки якому було зареєстровано нового користувача;

– password – пароль користувача;

– google_id – Google ID користувача використовується при реєстрації в додатку через Google;

– facebook_id – Facebook ID користувача використовується при реєстрації в додатку через Facebook.

Roles:

– name – ім'я ролі (для даного стовпця використано формат enum (перелічуваний тип) тому що адмін може створити лише певні ролі).

Role_user:

– role_id – primary id з таблиці “roles”, що використовується для відношення між таблицями “users” та “roles”;

– user_id – primary id з таблиці “users”, що використовується для відношення між таблицями “users” та “roles”.

Products:

– name – ім'я продукту;

– cost – вартість продукту;

– amount – доступна кількість продукту;

– rating – рейтинг продукту;

– meta_title – заголовок продукту;

– description – опис продукту;

– content – вміст продукту, наприклад: матеріал, довжину, ширину, колір, гендер, тощо (для даного стовпця використано формат json, оскільки контент продукту може бути різного обсягу, а не фіксованого, має сенс зробити колонку типу json, щоб зберігати в колонці динамічні дані);

– images_content – зображення продукту (для даного стовпця використано формат json оскільки картинок продукту може бути достатньо мого, має сенс зробити колонку типу json, щоб зберігати в колонці динамічні дані).

Product_statuses:

– name – ім'я статусу.

Product_categories:

– name – ім'я категорії.

Product_categories_linked:

– product_id – primary id з таблиці “products”, використовується для відношення між таблицями “products” та “product_categories”;

– product_category_id – primary id з таблиці “product_categories”, використовується для відношення між таблицями “products” та “product_categories”.

Orders:

– user_id – primary id з таблиці “users”, використовується для відношення між таблицями “orders” та “users”;

– order_status_id – primary id з таблиці “order_statuses”, використовується для відношення між таблицями “orders” та “order_statuses”;

– coupon_id – primary id з таблиці “coupons”, використовується для відношення між таблицями “coupons” та “orders”;

– grand_total – підсумкова ціну замовлення;

– shipping_cost – ціна доставки;

– billing_email – email адрес покупця;

– billing_phone – номер телефону покупця;

– billing_city – місто проживання покупця;

– billing_street – вулиця проживання покупця;

– billing_house – номер будинку покупця.

Order_statuses:

– name – ім'я статусу (для даного стовпця використано формат enum (перелічуваний тип) тому що адмін може створити лише певні статуси).

Order_products:

– product_id – primary id з таблиці “products”, використовується для відношення між таблицями “orders” та “products”;

– order_id – primary id з таблиці “orders”, використовується для відношення між таблицями “orders” та “products”;

– coupon_id – primary id з таблиці “coupons”, використовується для відношення між таблицями “coupons” та “order_products”;

– quantity – кількість придбаного товару.

Referral_links:

– code – унікальний згенерований хеш код;

– user_id – primary id з таблиці “users”, використовується для відношення між таблицями “users” та “referral_links”;

– referral_program_id – primary id з таблиці “referral_programs”, використовується для відношення між таблицями “referral_programs” та “referral_links”.

Referral_programs:

– name – ім'я реферальної програми;

– uri – посилання реферальної програми;

– lifetime_minutes – час, скільки реферальна програма дійсна;

– reward – кількість балів, що отримає користувач при виконанні реферальної програми.

Referral_program_steps:

– name – ім'я кроку реферальної програми;

– goal – кількість очок, які потрібно набрати;

– referral_program_id – primary id з таблиці “referral_programs”, використовується для відношення між таблицями “referral_programs” та “referral_program_steps”;

– reward – кількість балів, що отримає користувач при виконанні кроку реферальної програми.

Referral_program_user_steps:

– program_id – primary id з таблиці “referral_programs”, використовується для відношення між таблицями “referral_programs” та “referral_program_user_steps”;

– step_id – primary id з таблиці “referral_program_steps”, використовується для відношення між таблицями “referral_program_steps” та “referral_program_user_steps”;

– user_id – primary id з таблиці “users”, використовується для відношення між таблицями “users” та “referral_program_user_steps”.

Coupons:

– name – ім'я купона;

– discount – значення знижки, що дає дисконт;

– cost – ціна купону;

– expires_at – час через який купон буде недійсним (цей стовпець має тип timestamp, щоб вказати точне значення часу).

Coupon_user:

– user_id – primary id з таблиці “users”, використовується для відношення між таблицями “users” та “coupon_user”;

– coupon_id – primary id з таблиці “coupons”, використовується для відношення між таблицями “coupons” та “coupon_user”;

– available – значення, яке показує, чи доступний купон для користувача (ця колонка має тип tinyint, щоб зберігати булеан значення);

– status – статус юзер купона (для даного стовпця використано формат enum (перелічуваний тип) тому що тільки певний статус може бути для купона користувача).

Як можна бачити у сутностей “users” і “roles” зв'язок багато до багатьох, є зв'язуюча таблиця (role_user), у користувача може бути кілька ролей, а в свою чергу у ролі може бути кілька користувачів.

Зв'язок один до багатьох між таблицями “product_statuses” та “products”, у товара може бути тільки один статус, а в свою чергу один і той же статус може бути у багатьох товарів.

Зв'язок між таблицями “product_categories” та “products” є M:M, оскільки в одного і того ж товару може бути кілька категорій, і в однієї і тієї ж категорії товару може бути кілька продуктів.

Зв'язок між таблицями “order_statuses” та “orders” – один до багатьох, у замовлення може бути тільки один статус, а в свою чергу один і той же статус може бути у багатьох замовлень.

Зв’язок між таблицями “referral_links” та “users” – один до багатьох, у реферального посилання може бути лише один користувач, оскільки вона унікальна, але у свою чергу у користувача може бути кілька реферальних посилань.

Зв’язок між таблицями “referral_links” та “referral_programms” – один до багатьох, у реферального посилання може бути лише одна реферальная програма, але у свою чергу у реферальної програми може бути кілька реферальних посилань.

Зв'язок між таблицями “products” та “orders” є M:M, оскільки той самий товар може бути в декількох замовленнях.

Детальніше розглянемо ключові таблиці.

Таблиця “users” (див. рисунок 2.4), зберігає інформацію про користувачів системи, та і має наступні поля.

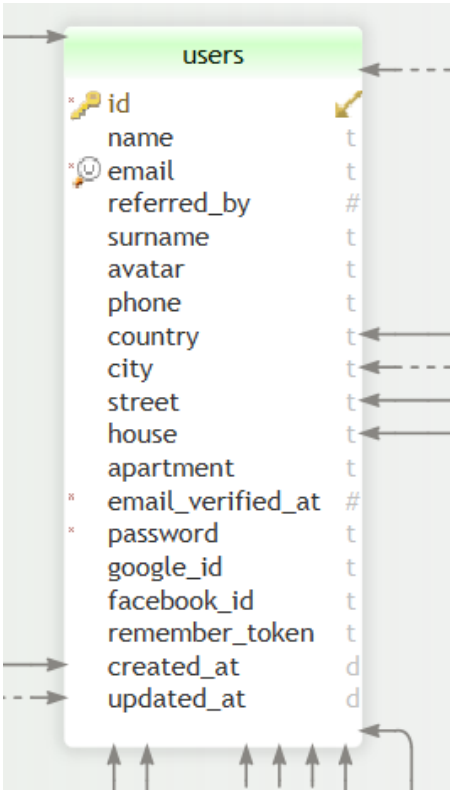

Рисунок 2.4 – Структура таблиці users

Таблиця “roles” (див. рисунок 2.5), зберігає інформацію про доступні ролі інформаційної системи.

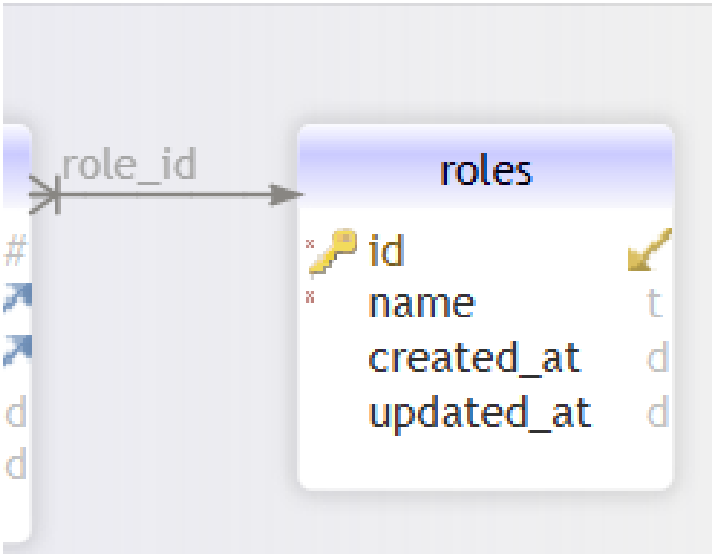

Рисунок 2.5 – Структура таблиці roles

Таблиця “role_user” (див. рисунок 2.6), є сполучною таблицею для таблиць “roles” та “users”.

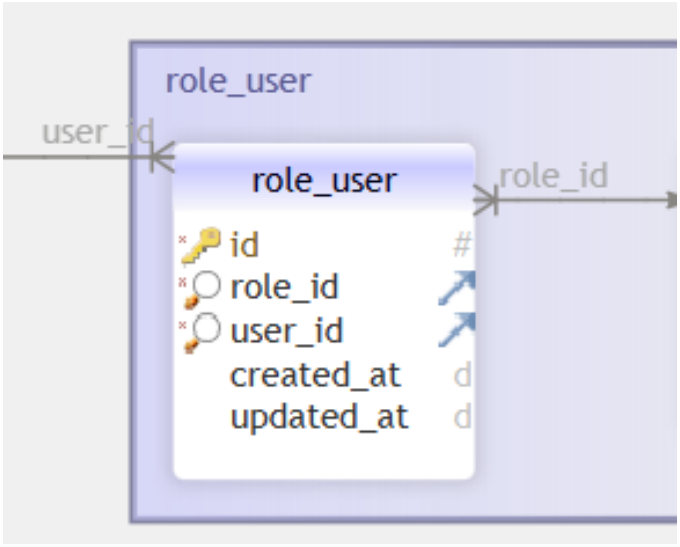

Рисунок 2.6 – Структура таблиці role_user

Таблиця “orders” (див. рисунок 2.7) зберігає дані про оформлені замовлення конкретного користувача, а саме id користувача, id статусу замовлення, id купону (якщо використовувався) і тощо.

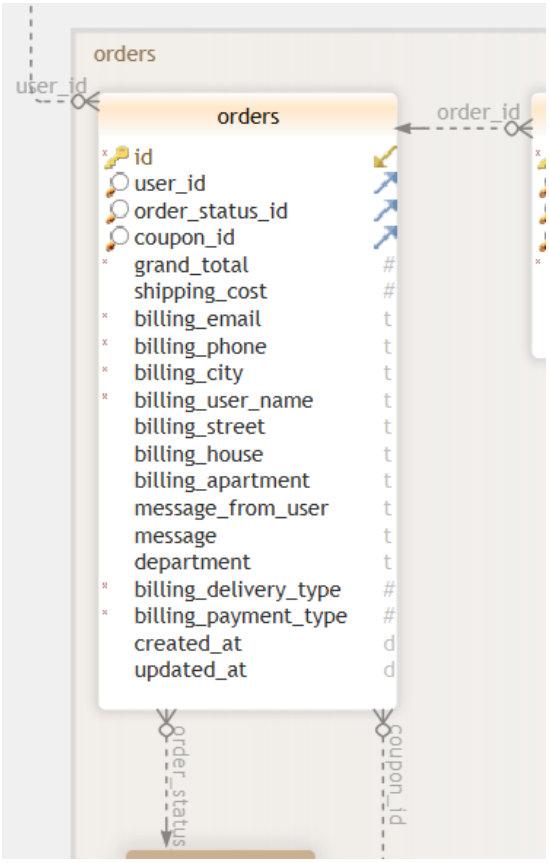

Рисунок 2.7 – Структура таблиці orders

Таблиця “order_products” (див. рисунок 2.8) є сполучною таблицею для таблиць “orders” і “products”.

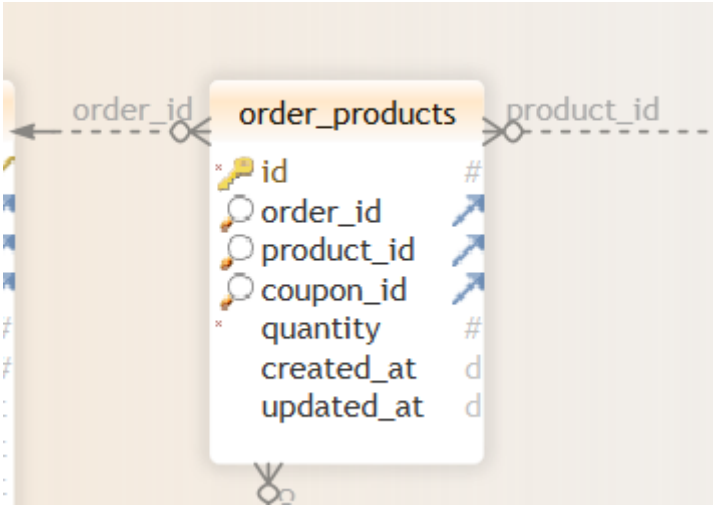

Рисунок 2.8 – Структура таблиці order_products

Таблиця “referral_links” (див. рисунок 2.9) зберігає інформацію про реферальні посилання.

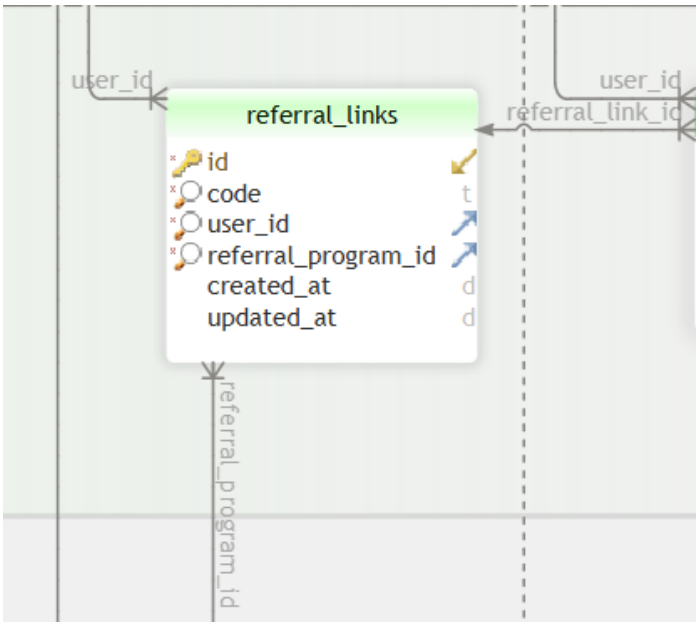

Рисунок 2.9 – Структура таблиці referral_links

Таблиця “referral_programs” (див. рисунок 2.10) зберігає інформацію про реферальні програми.

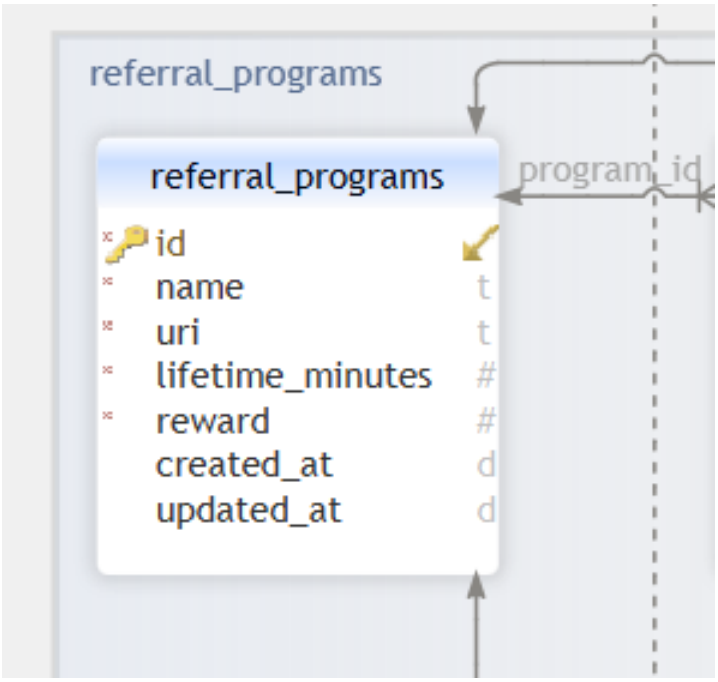

Рисунок 2.10 – Структура таблиці referral_programs

Таблиця “coupons” (див. рисунок 2.11) зберігає інформацію щодо купонів додатка.

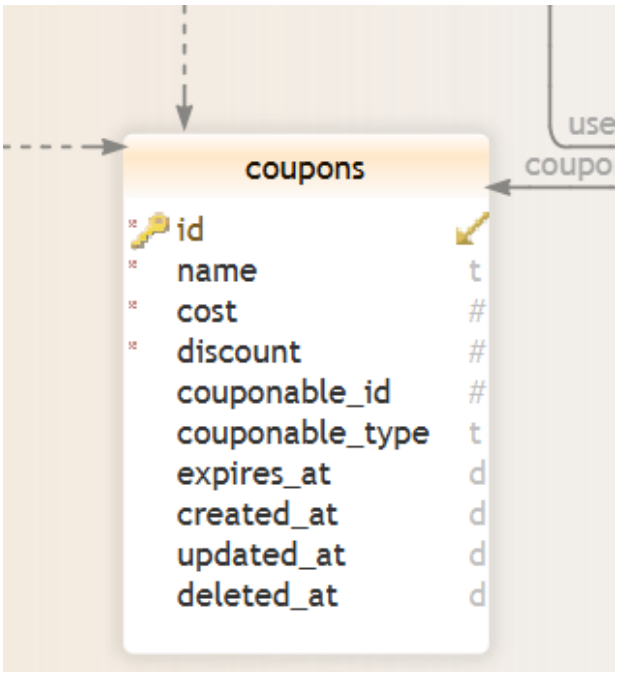

Рисунок 2.11 – Структура таблиці coupons

Таблиця “coupon_user” (див. рисунок 2.12) ставить асоціацію купона з обліковим записом користувача.

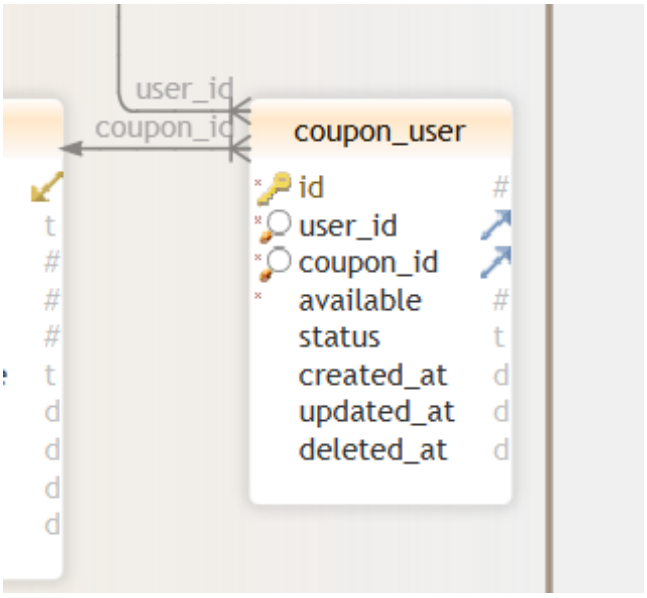

Рисунок 2.12 – Структура таблиці coupon_user

Таблиця “products” (див. рисунок 2.13) зберігає основну інформацію про продукти, а саме: ім'я, ціну, доступну кількість, опис, рейтинг продукту тощо.

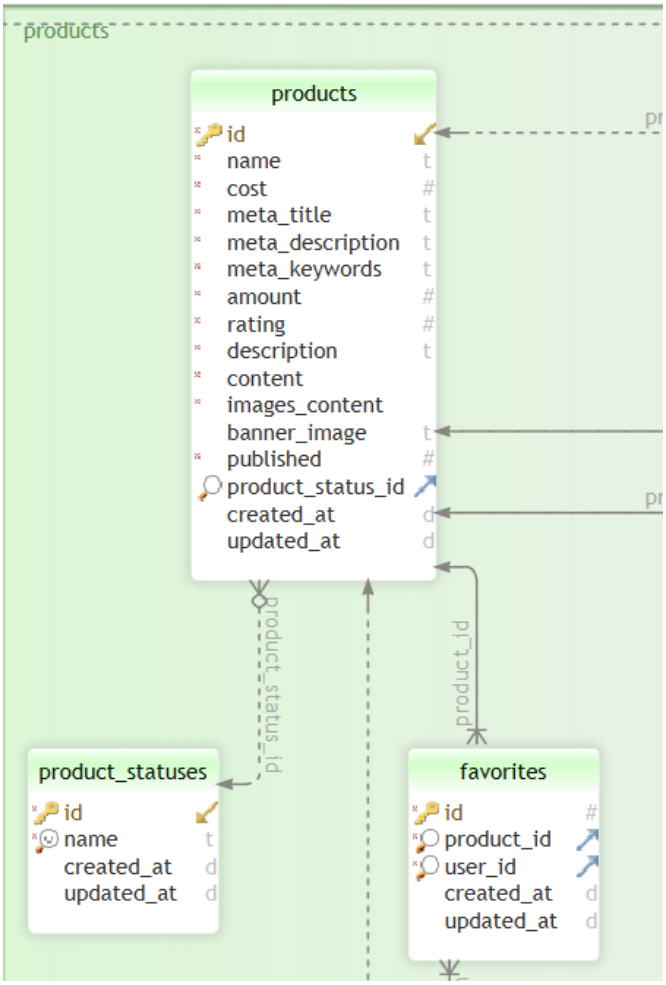

Рисунок 2.13 – Структура таблиці products

2.2 Діаграма прецедентів

Діаграма прецедентів (use case diagram) в UML – діаграма, що відображає відносини між акторами і прецедентами і є складовою частиною моделі прецедентів, що дозволяє описати систему на концептуальному рівні.

Можна виділити як мінімум 3 актора (сутності), який виконують роль в даній системі, а саме покупець, інтернет магазин, і оператор.

В якості основного актора описуваної системи можна розглядати відвідувача інтернет-магазину, який може переглядати список товарів, розміщувати товар у віртуальний кошик і змінювати його вміст. Відвідувач може стати покупцем, якщо він приймає рішення про оформлення замовлення на купівлю обраних ним товарів. Іншими акторами даної системи можуть виступати менеджер інтернет магазину і бухгалтер. При цьому менеджер може змінювати список товарів і специфікувати умови для надання бонусної знижки, а бухгалтер – приймати плату за вибраний покупцем товар.

У свою чергу, взаємодія користувача з інтернет магазином може відстежуватися в інформаційній системі. Наприклад, інформаційна система може записувати в базу даних замовлення користувача, його список бажаного. обробляти замовлення, тощо. Слід зазначити, що інформаційна система бере на себе відповідальність за обробку інформації, роботу із замовленнями, дозволяє адміністраторам гнучко керувати контентом.

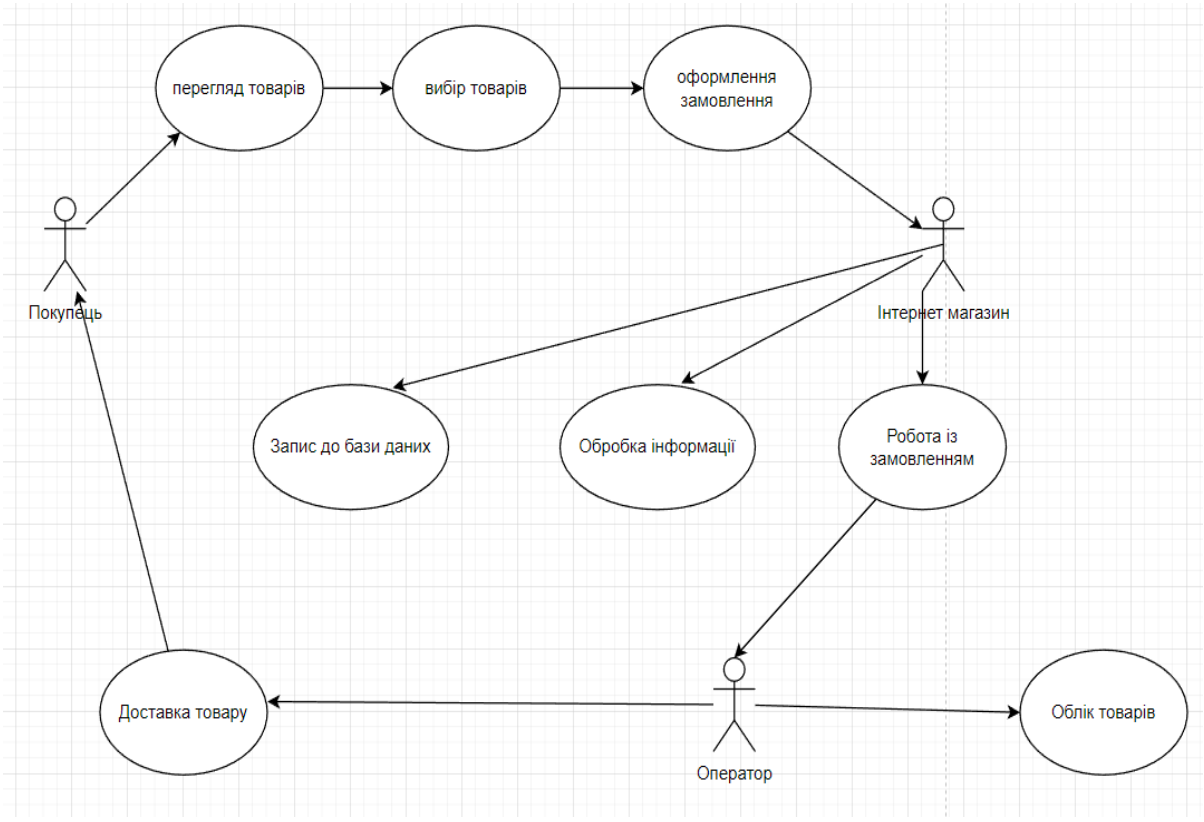

Рисунок 2.14 – Діаграма прецедентів

Однак, найбільш значущим прецедентом для цієї системи і її акторів являється прецедент Замовлення товарів. Для нього побудуємо додаткову діаграму прецедентів, що пояснює цей варіант використання.

Рисунок 2.15 – Діаграма варіантів використання основного прецеденту Замовлення товарів

2.3 Діаграма діяльності

Діаграма діяльності – UML-діаграма, що дозволяє описувати логіку процедур, бізнес-процеси і потоки робіт. У багатьох випадках вони нагадують блок-схеми, але принципова різниця між діаграмами діяльності і нотацією блок-схем полягає в тому, що перші підтримують паралельне процеси.

Ця діаграма дозволяє будь-кому, хто виконує цей процес, вибирати порядок дій. Іншими словами, діаграма тільки встановлює правила обов'язкової послідовності дій, яким користувач повинен слідувати. Це важливо для моделювання бізнес-процесів, оскільки ці процеси часто виконуються паралельно. Такі діаграми також корисні при розробці

паралельних алгоритмів, в яких незалежні потоки можуть виконувати роботу паралельно.

Діаграми діяльності, так само як і блок-схеми, – загальний та потужний засіб опису алгоритмів. Їх з успіхом також можна застосовувати для моделювання поведінки людей, пристроїв і організацій при виконанні бізнеспроцесів.

Ось наприклад приведена діаграма що побудовану на прикладі процесу обробки звернення клієнта до менеджера інтернет-магазину і подальшого оформлення замовлення.

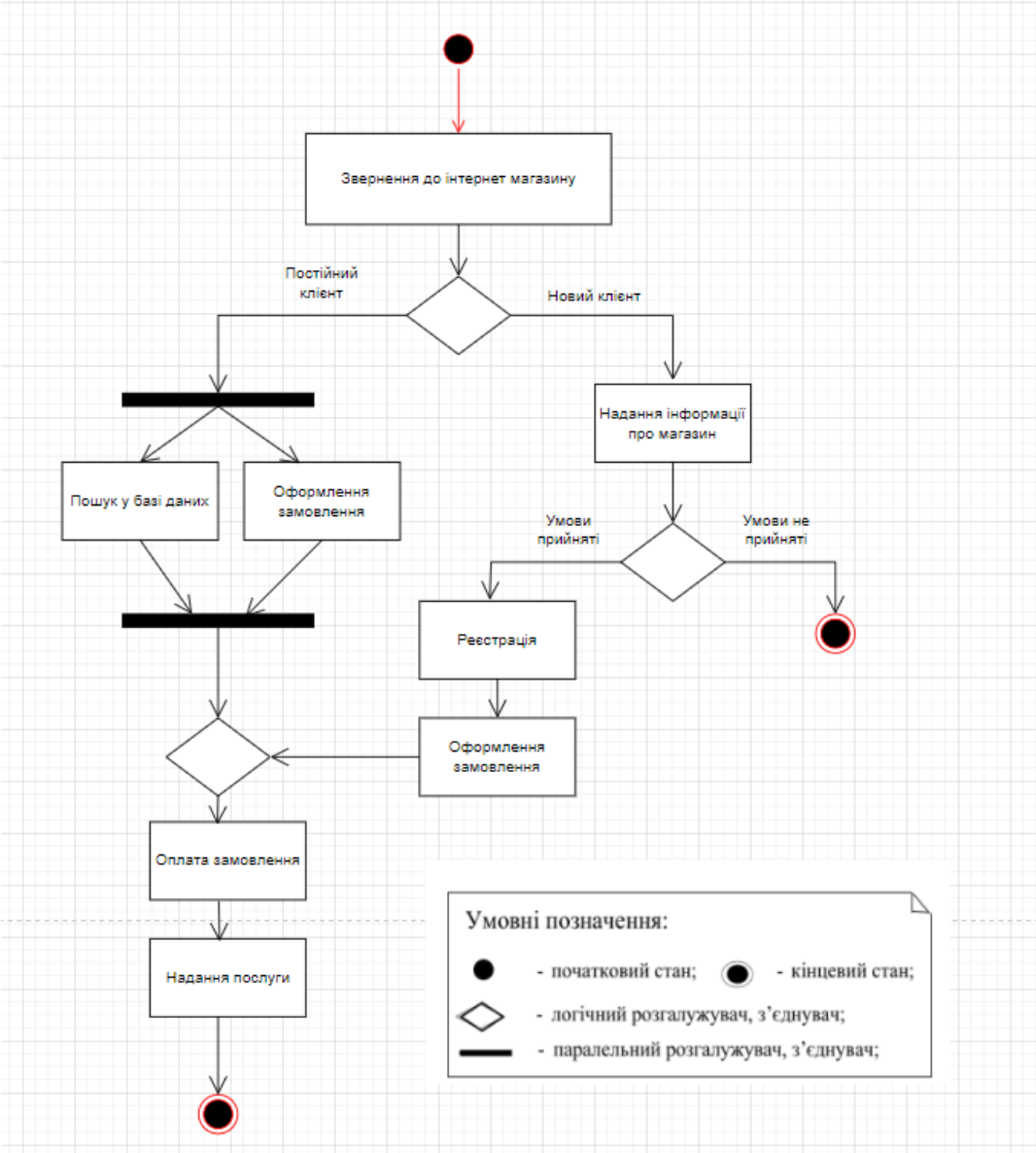

Рисунок 2.16 – Діаграма діяльності

Докладніше розглянемо цю диграму та опишемо її дії. Спершу користувач звертається в інтернет магазин, далі йде логічний розгалужувач, де перед користувачем стоїть питання чи є він постійним клієнтом або він новий клієнт.

Розглянемо перший варіант, коли користувач є новим клієнтом. Користувачеві надається інформація про інтернет магазин, потім йому потрібно вирішити, чи хоче він купувати щось в даному інтернет магазині, якщо ні – процес завершений, якщо так – процес реєстрації, оформлення замовлення, потім логічний з'єднувач, що призводить до етапу оплати замовлення і в кінцевому підсумку до отримання послуги та завершення процесу.

Розглянемо другий варіант, коли користувач є постійним клієнтом та хоче оформити замовлення. На даному етапі, на діаграмі можна спостерігати розпаралелювання потоків, а потім злиття, це говорить про те, що дві дії, а саме пошук у базі даних та оформлення замовлення можуть відбуватися одночасно. Потім йде логічний з'єднувач (так само як і в першому варіанті), що призводить до етапу оплати замовлення і в кінцевому підсумку до отримання послуги та завершення процесу.

Також варто відзначити, що операції можуть бути розбиті на вкладені діяльності (subactivities) та окремі дії, з'єднані між собою потоками, що йдуть від виходів одного вузла до входів іншого. Занадто великі діаграми діяльності часом складні розуміння. Тому їх структуровані розділи можуть бути організовані як підлеглих діяльностей – особливо у випадках, коли такі виконуються неодноразово межах головної. При цьому головна та підлеглі діяльності зображуються на окремих діаграмах. Саме тому можна розбити діаграму діяльності на розділи (partitions), щоб показати, хто що робить, тобто які операції виконує той чи інший клас або підрозділ підприємства. Наприклад детальніше розберемо розглянемо алгоритм доставки.

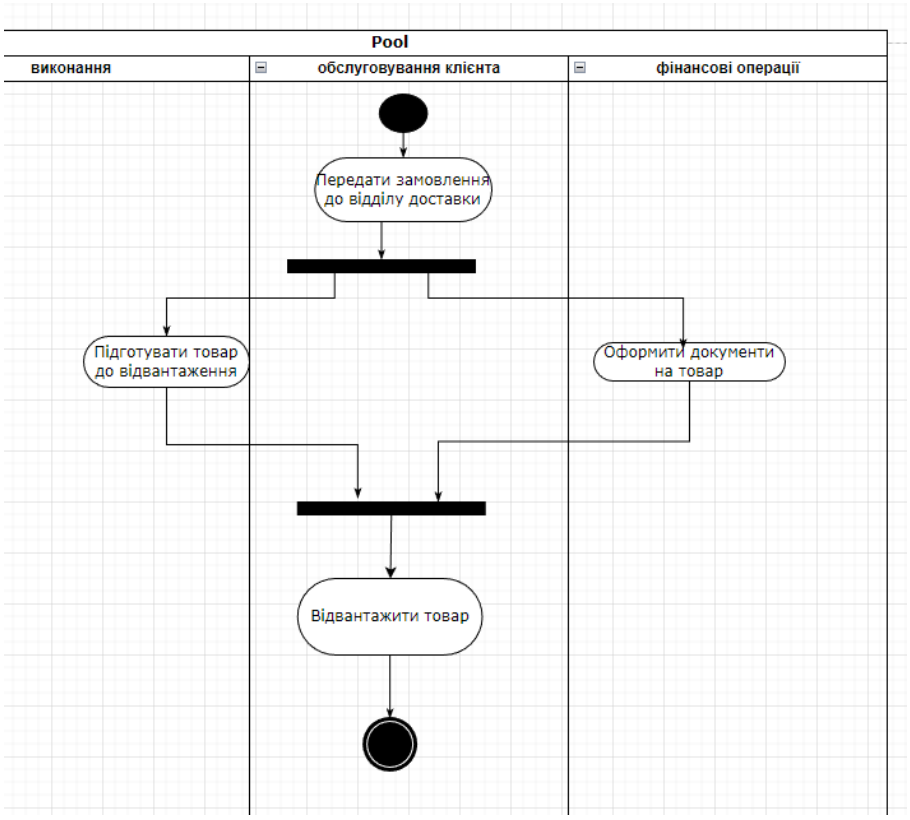

Рисунок 2.17 – Діаграма діяльності, розбита на вкладені діяльності

Дана діаграма показує процес оформлення замовлення, можна помітити, що на діаграмі присутнє розпаралелювання потоків, а потім злиття, це говорить про те, що такі процеси як підготовка товару до відвантаження та оформлення документів на товар можуть відбуватися одночасно, але в кінцевому підсумку, кінцевим етапом виступає відвантаження товару.

2.4 Діаграма компонентів

Діаграма компонентів описує особливості фізичного представлення системи. Діаграма компонентів дозволяє визначити архітектуру розроблюваної системи, встановивши залежності між програмними компонентами, в ролі яких може виступати вихідний, бінарний і виконуваний код. У багатьох середовищах розробки модуль або компонент відповідає файлу.

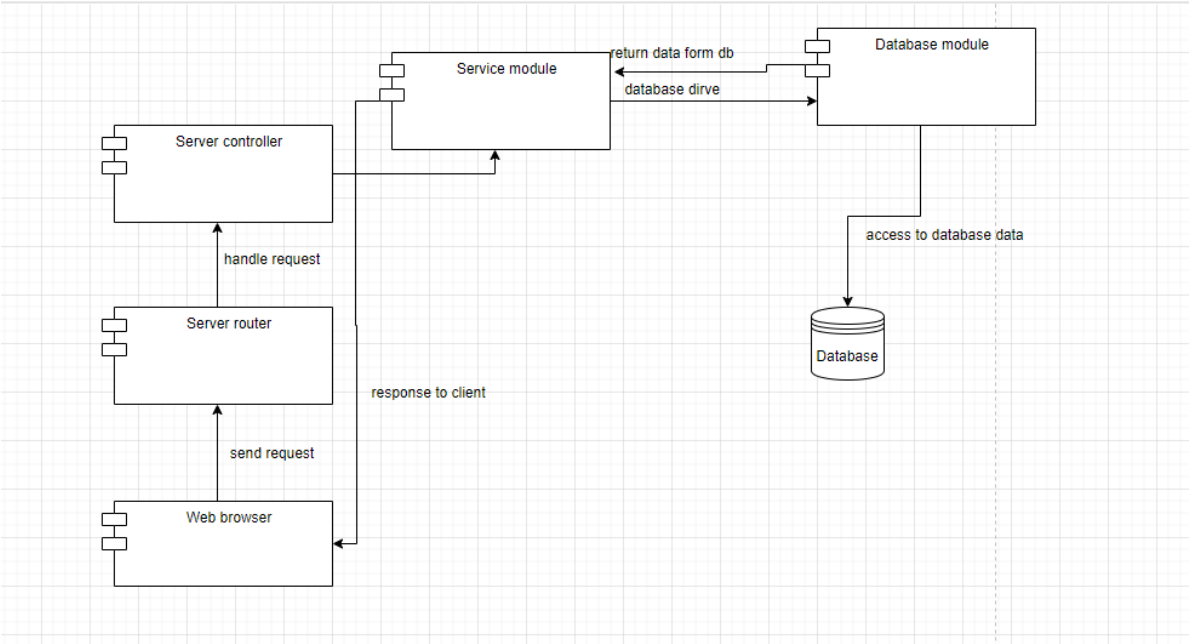

Рисунок 2.18 – Діаграма компонентів

На наведеному малюнку, веб браузер може взаємодіяти з бек енд роутерами, посилаючи на них запити, в свою чергу роутери будуть перенаправляти запити до контролера, де буде відбувається обробка запиту та проводитися маніпуляції з базою даних.

2.5 Діаграма класів

Діаграма класів займає центральне місце в проектуванні об'єктно-орієнтованої системи. Нотація класів використовується на різних етапах проектування і будується з різним ступенем деталізації. Мова UML застосовується не тільки для проектування, але і з метою документування, а також ескізування проекту. На рис. 2.17 наведено концептуальну діаграму класів, засновану на аналізі предметної області. Діаграма визначає структуру системи, показуючи її класи, їх атрибути та методи.

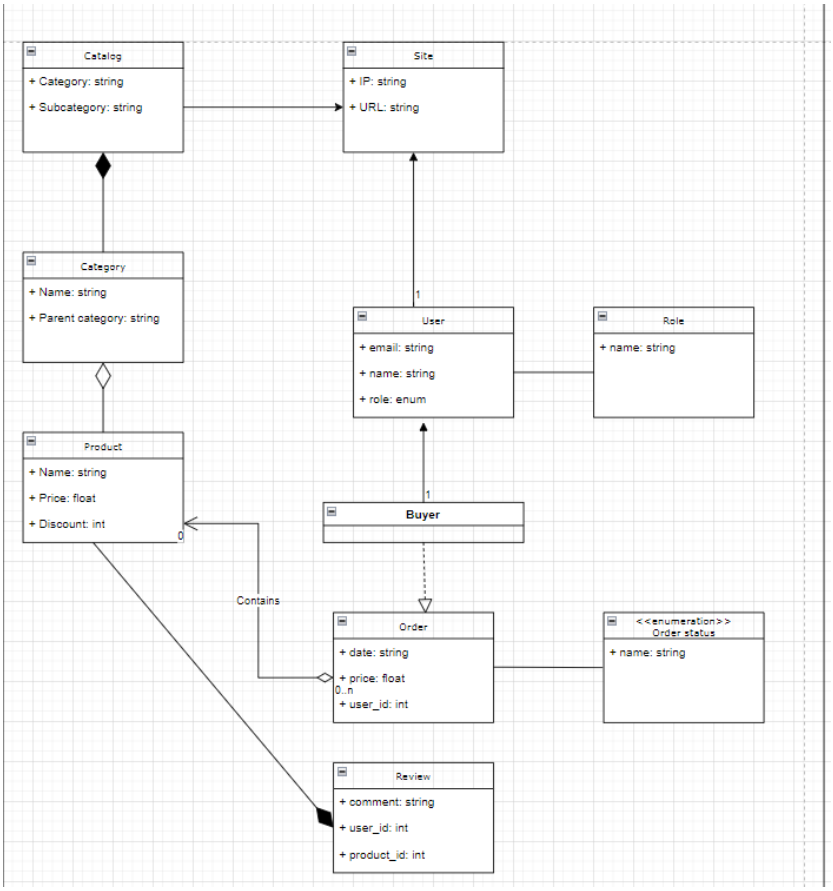

Рисунок 2.19 – Діаграма класів

Опишемо докладніше елементи діаграми. Каталог містить у собі продукти, але екземпляри продуктів можуть існувати, але при цьому не знаходиться в якійсь категорії, тому тип зв'язку між цими класами – агрегація.

Примірник класу характеристика продукту існує, доки існує продукт, коли продукт буде видалений, всі пов'язані характеристики також будуть видалені, тому тип зв'язку композиція (одна колекція не може існувати без іншої).

Примірники класу відгук не можуть існувати без продукту, до якого був залишений відгук, тому тип зв'язку композиція.

Клас “Статуси замовлення” зберігає можливі статуси замовлення, наприклад: надіслано, обробляється, отримано і т.д. На діаграмі представлені як переліки.

2.6 Діаграма розгортання

Діаграма розгортання представляє фізичне розташування системи, показуючи, на якому фізичному обладнанні запускається та чи інша складова програмного забезпечення. Діаграма розгортання дуже проста.

Головними елементами діаграми є вузли, пов'язані інформаційними шляхами. Вузол (node) – це те, що може містити програмне забезпечення.

Вузли можуть містити артефакти (artifacts), які є фізичним уособленням програмного забезпечення.

Такими артефакт файлами можуть бути виконувані файли (такі як файли .exe, виконавчі файли, файли DLL, файли JAR, складання або сценарії) або файли даних, конфігураційні файли, HTML-документи, JS файли, тощо.

Наприклад, якщо ми говоримо про фронт енд частину веб додатку, то для розвороту буде потрібно лише створити білд коду, який містить уже все необхідне для роботи програми. Тобто білд код і виступатиме артефактом, що несе у собі додаток. Також не варто забувати, що для серву білд коду буде потрібно налаштувати сервер, наприклад Apache або Nginx.

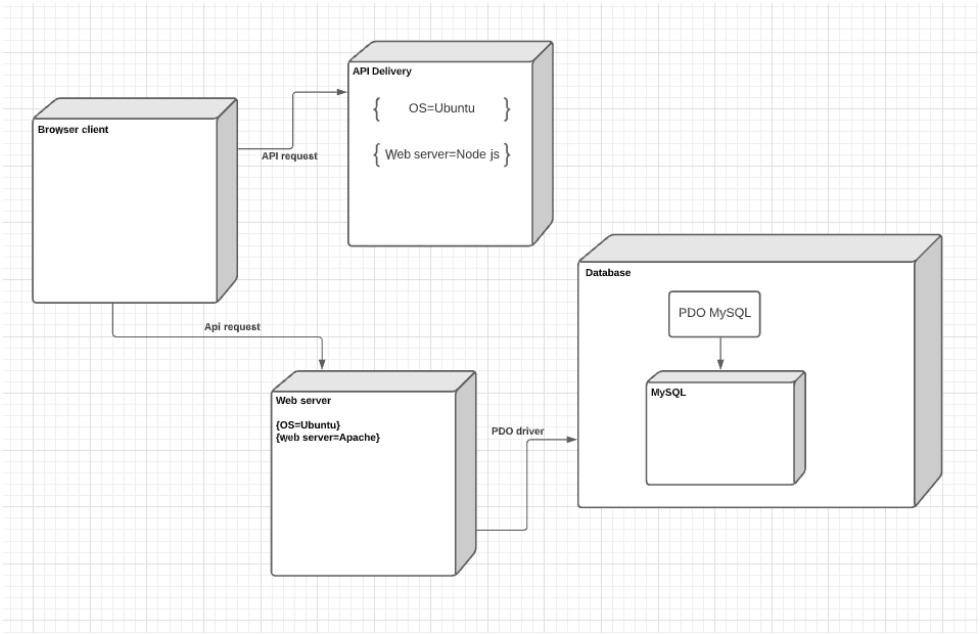

Рисунок 2.20 – Діаграма розгортання

3 ПРОГРАМУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Тестування бекенд АПІ

Гарною практикою є покрити функціонал API тестами, описавши базові тест кейси. Тест кейс – це перевірка працездатності програми чи проекту. Написати тест кейс означає створити текстовий опис процесу тестування якоїсь частини або функції проекту. Тест кейси потрібні, щоб члени команди могли перевірити програму та познайомитись з нею, не читаючи весь код, а вивчивши лише тест кейс. Тестування допомагає переконатися, що програма виконує поставлену перед нею мету та зможе коректно взаємодіяти з іншими програмами. На прикладі розглянемо тести для наступного функціоналу: реєстрація акаунту, логін в систему, а також тести для операцій CRUD пов'язані з сутністю продукту.

Спочатку напишемо базовий тест для успішної реєстрації. Оскільки використовується Laravel фреймворк, можно описати фабрику для моделі User, що спростить тестування надалі. Для початку треба пояснити що таке фабрика і навіщо вона потрібна. Під час тестування може знадобитися вставити кілька записів у базу даних перед виконанням тесту. Замість того, щоб вручну вказувати значення кожного стовпця під час створення цих тестових даних, Laravel дозволяє визначити стандартний набір атрибутів для кожної з ваших моделей Eloquent за допомогою фабрик моделей. Після того, як ми описали фабрику для нашої моделі, ми робимо запит на нашу API, передавши дані. Далі ми хочемо перевірити 3 моменти. У перших статус відповіді повинен бути 201, потім перевірити структуру бекенд відповіді, а також переконається, що наш зареєстрований користувач дійсно присутній у базі даних.

```php
public function testSuccessfulRegistration() {
    $user = User::factory()->makeOne([
        'password' => bcrypt($password = 'i-love-laravel'),
    ]);

    $response = $this->post( $this->baseUrl, [
        'name' => $user->name,
        'email' => $user->email,
        'password' => $password,
    ]);

    $response->assertStatus( status: Response::HTTP_CREATED);
    $response->assertExactJson(
        [
            "success" => true,
            'data' => [
                'user_id' => $response['data']['user_id'],
            ],
            "message" => "Registered successfully"
        ]
    );
}
```

Рисунок 3.1 – Приклад тестування успішної реєстрації

Тут ще необхідно звернути увагу на те, що в тесті використовується метод фабрики makeOne, даний метод створює за нас екземпляр моделі з усіма потрібними полями.

```php
class UserFactory extends Factory
{
    /** The name of the factory's corresponding model. ...*/
    protected $model = User::class;

    /** Define the model's default state. ...*/
    public function definition()
    {
        return [
            'name' => $this->faker->name,
            'surname' => $this->faker->lastName,
            'email' => $this->faker->unique()->safeEmail,
            'email_verified_at' => 0,
            'phone' => substr($this->faker->e164PhoneNumber,  offset: 1),
            'city' => $this->faker->city,
            'street' => $this->faker->streetName,
            'house' => $this->faker->buildingNumber,
            'apartment' => $this->faker->buildingNumber,
            'password' => $this->faker->text( maxNbChars: 10),
            'remember_token' => null,
        ];
    }
```

Рисунок 3.2 – Приклад фабрики для моделі User

Також варто створити тест, який би перевіряв усі поля передані АПА на валідацію. Напишемо тест, який має очікувати 422 статус і повертати повідомлення про не валідні поля.

```php
protected string $baseUrl = '/api/v1/client/sign-up';

public function testRequiredFieldsForRegistration() {
    $this->json( method: 'post', $this->baseUrl, [])
        ->assertStatus( status: Response::HTTP_UNPROCESSABLE_ENTITY)
        ->assertExactJson([
            "success" => false,
            "error" => [
                "message" => [
                    "password" => ["The password field is required."],
                    "email" => ["The email field is required."],
                    "name" => ["The name field is required."],
                ],
                "details" => "Validation Error.",
            ]
        ]);
}
```

Рисунок 3.3 – Приклад тестування валідації для реєстрації

Ще важливим моментом є перевірка на те, що один і той же користувач не може бути зареєстрований двічі, оскільки email є унікальним.

Для цього тесту ми можемо скористатися методом createOne, який надає фабрика моделі, що трохи спростить нам завдання. Метод createOne створить екземпляр моделі з усіма потрібними полями та помістить його в базу даних за нас.

Варто зазначити, що головною відмінністю між методом createOne та makeOne є те, що метод makeOne створює екземпляр моделі без її збереження в базі даних. У той час як метод createOne створює екземпляр моделі та зберігає їх у базі даних за допомогою методу збереження Eloquent.

Використання методу createOne дозволяє уникнути ручного внесення даних нового користувача до бази даних, наприклад, використовуючи метод реєстрації.

```php
public function testFailedCannotCreateTheSameUserAgain() {
    $fakeUser = User::factory()->createOne();

    $response = $this->post( $this->baseUrl, [
        'name' => $fakeUser->name,
        'email' => $fakeUser->email,
        'password' => "testpassword",
    ]);
    $response->assertStatus( status: Response::HTTP_UNPROCESSABLE_ENTITY);
    $response->assertExactJson(
        [
            "success" => false,
            "error" => [
                "message" => [
                    "email" => ["The email has already been taken."],
                ],
                "details" => "Validation Error.",
            ]
        ]
    );
}
```

Рисунок 3.4 – Приклад тестування емейла на унікальність під час реєстрації

Далі напишемо тест кейси для логіну. Для початку напишемо успішний тест логіну, який має перевірити статус відповіді, який має бути 200 у разі успішного логіну в система, а також переконаємось у структурі відповіді.

```php
protected string $baseUrl = '/api/v1/client/sign-in';

public function testSuccessfulLogin() {
    $user = User::factory()->createOne([
        'password' => bcrypt($password = 'i-love-laravel'),
    ]);

    $this->json( method: 'post', $this->baseUrl, [
        "email" => $user->email,
        "password" => $password,
    ])
        ->assertStatus( status: Response::HTTP_OK)
        ->assertJsonStructure(
            [
                "success",
                "message",
                "data" => [
                    'logged_in_user_id',
                    'token'
                ]
            ]
        );
}
```

Рисунок 3.5 – Приклад тестування успішного логіну

Варто перевірити валідацію при логіні, для цього напишемо тест, який повинен буде повернути статус 422, а також дані про поля, що не пройшли валідацію.

```php
public function testRequiredPasswordAndEmailFieldsForLogin() {
    $this->json( method: 'post', $this->baseUrl, [])
        ->assertStatus( status: Response::HTTP_UNPROCESSABLE_ENTITY)
        ->assertExactJson([
            "success" => false,
            "error" => [
                "message" => [
                    "password" => ["The password field is required."],
                    "email" => ["The email field is required."],
                ],
                "details" => "Validation Error.",
            ]
        ]);
}
```

Рисунок 3.6 – Приклад тестування валідації для логіну

Необхідно перевірити, що даного користувача в системі не існує і надані дані користувачем не валідні.

```php
public function testFailedLogin()
{
    $payload = [
        "name" => "fake name",
        "email" => "fake@gmail.com",
        "password" => "fake_password",
    ];

    $this->json( method: 'post', $this->baseUrl, $payload)
        ->assertStatus( status: Response::HTTP_BAD_REQUEST)
        ->assertExactJson(
            [
                "success" => false,
                "error" => [
                    "details" => "Provide correct credentials",
                    "message" => "Bad Request"
                ],
            ]
        );
}
```

Рисунок 3.7 – Приклад тестування на неіснуючого користувача

Опишемо базові тести для сутності продукт, а саме: отримання всіх продуктів, отримання продукту по id, перевірка на те, що продукту за вказаним id не знайдено, продукт не може бути створений так як потрібна авторизація, продукт не може бути створений так як відсутні необхідні дозволи, успішне створення продукту, оновлення. видалення продукту. З усім вмістом розглянутих тестів можна буде ознайомитись у додатках.

Рисунок 3.8 – Приклад тестування дозволів на створення продукту

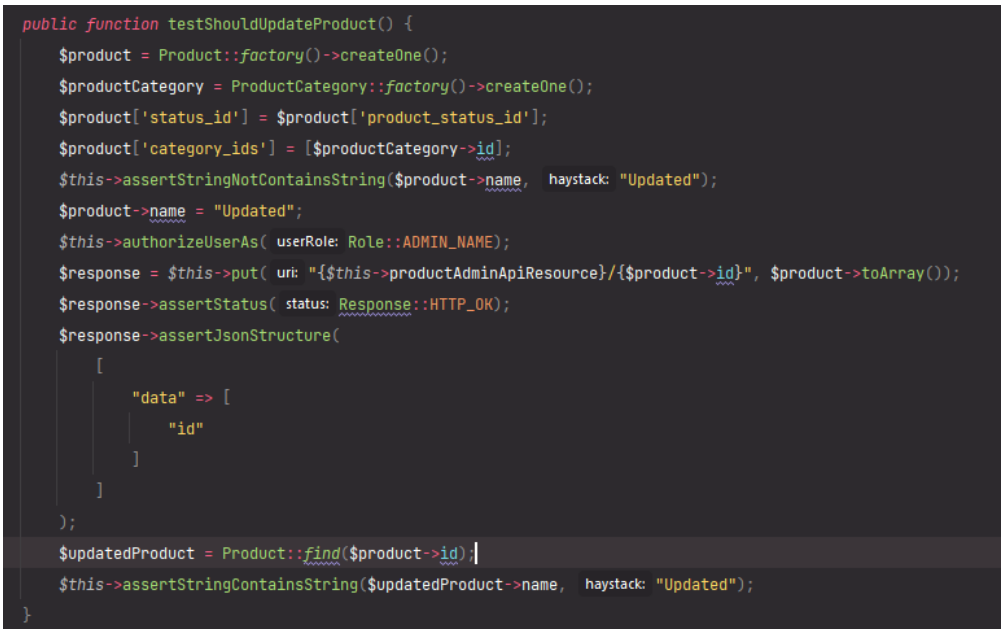

Рисунок 3.9 – Приклад тестування оновлення продукту

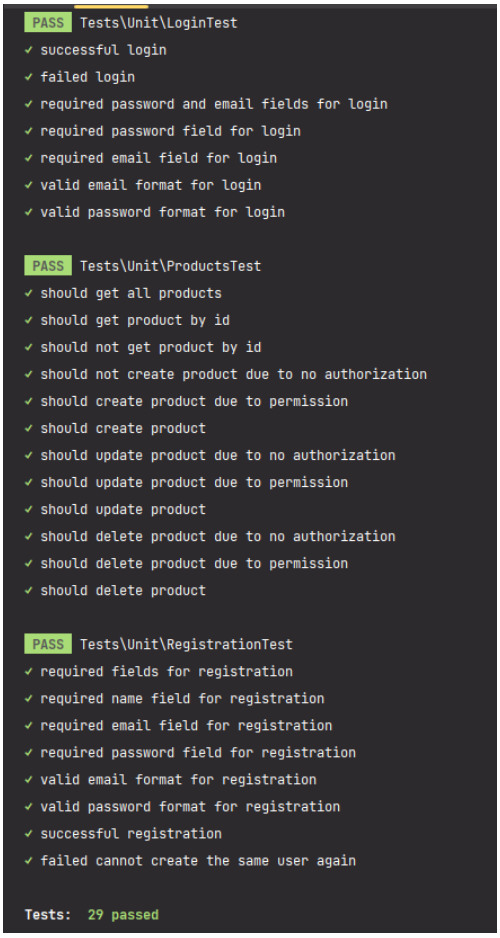

Рисунок 3.10 – Приклад проходження успішних тестів

Більш детально з тестами авторизації/реєстрації, сутності продукт можна ознайомитись у додатках А, Б та В.

3.2 Налаштування докер контейнера для бекенд частини

Докер спрощує процес розгортання програми, що дозволяє заощадити достатню кількість часу іншому розробнику, якому необхідно розгорнути проект.

Докер дозволяє запускати додаток з усіма необхідними модулями, ПО, сервісами в контейнері, що не вимагає установки всіх залежностей на призначеному для користувача ПК.

Також з його використанням можна не турбується за безпеку, завдяки контейнеризації, будь-який код, запущений в контейнері не може нашкодити основної операційної системи.

Напишемо докер файл для розвороту основних сервісів, таких як MySQL, Apache2, phpMyAdmin для можливості взаємодіяти з базою даних за допомогою призначеного для користувача інтерфейсу, а також загорнемо саме API додаток в контейнер, докладніше з конфігураційними файлами докера можна ознайомитись в додатках [8].

Після чого для запуску програми буде достатньо однієї лише команди: docker-compose up -d && docker-compose logs -f.

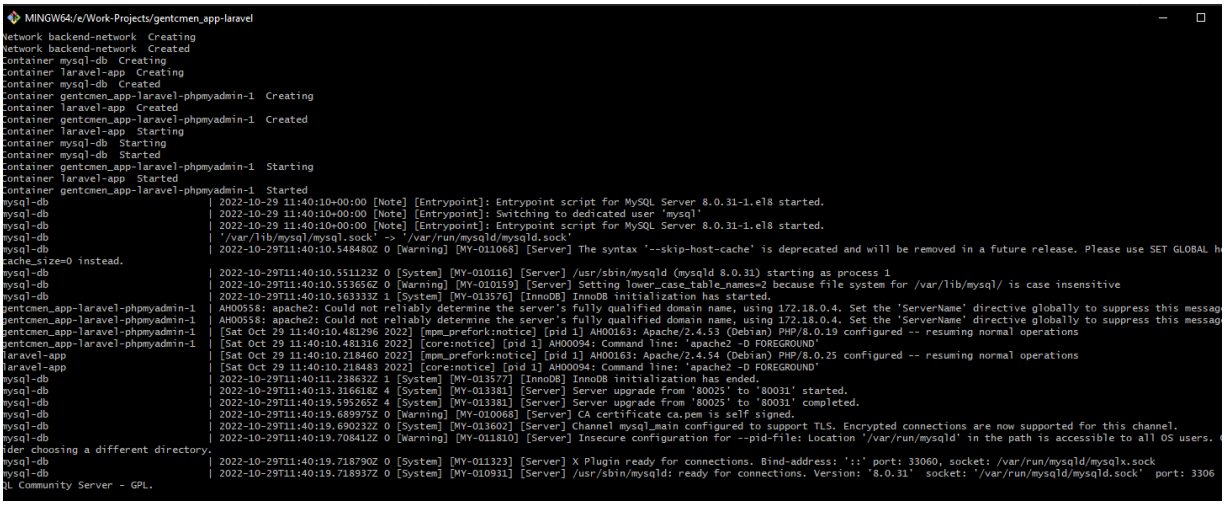

Рисунок 3.11 – Контейнеризація бекенд частини

Більш детально з конфігурацією Docker можна ознайомитись у додатках Г та Д.

3.3 Інтерфейс веб-додатку

Нижче наведені результати інтерфейсу розробленого веб додатка. Реалізовано базові сторінки для взаємодії з інтерфейсом.

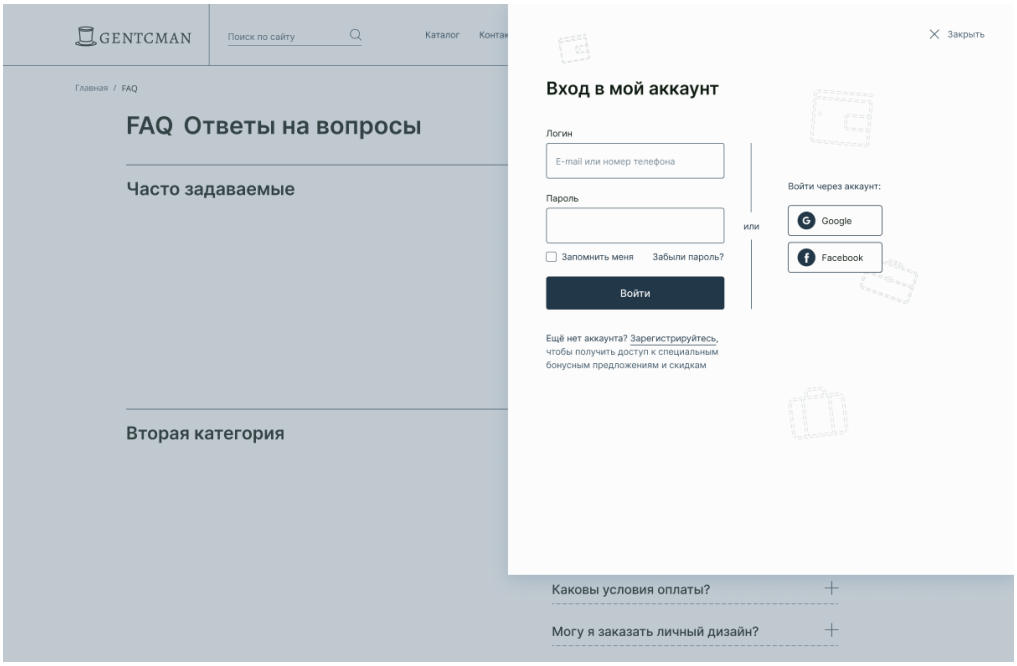

Рисунок 3.12 – Форма логіну

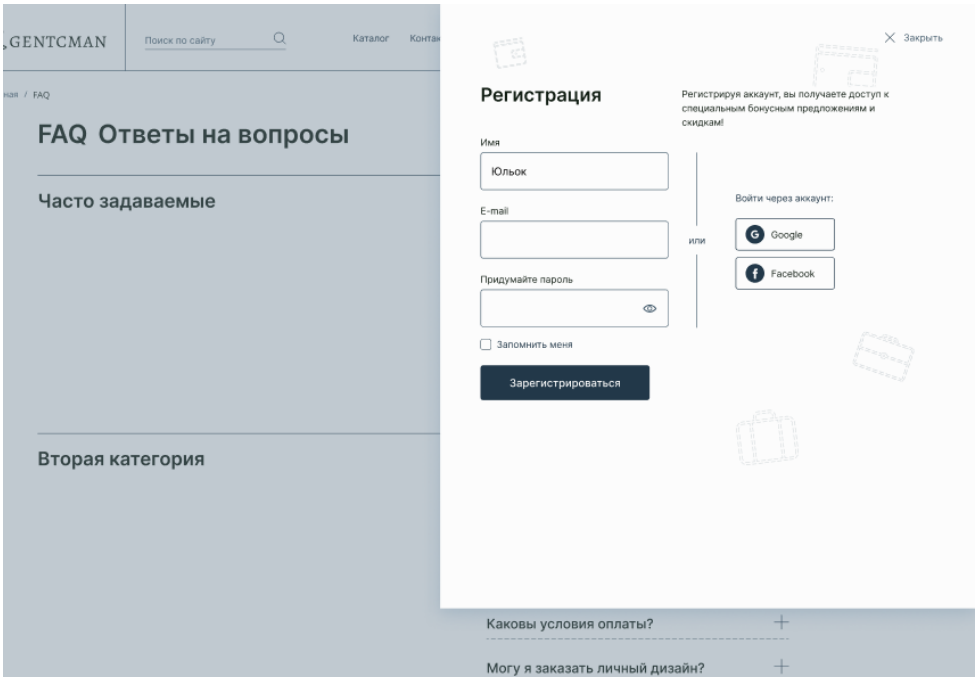

Рисунок 3.13 – Форма реєстрації

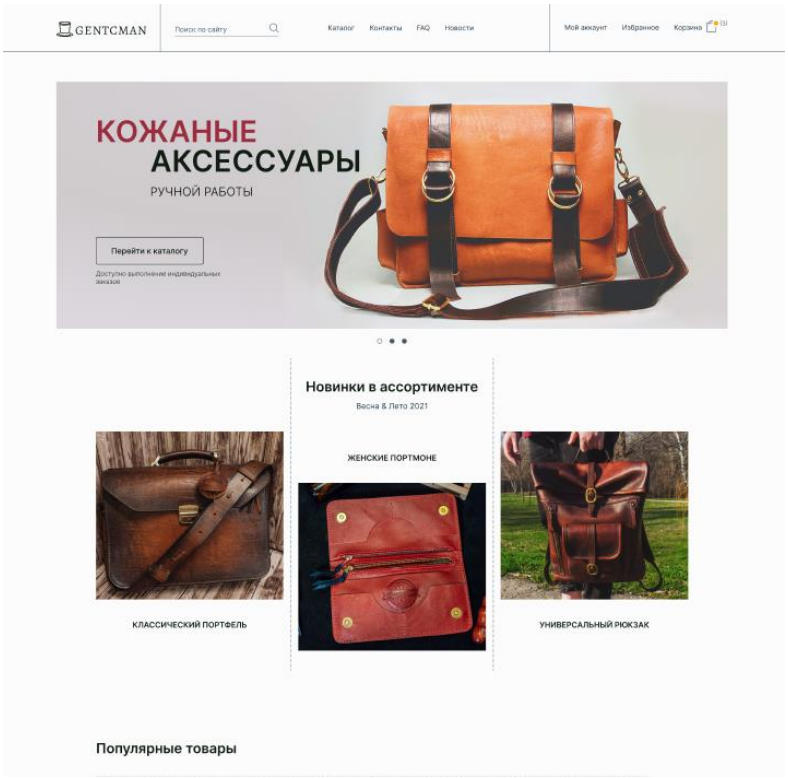

Рисунок 3.14 – Головна сторінка сайту

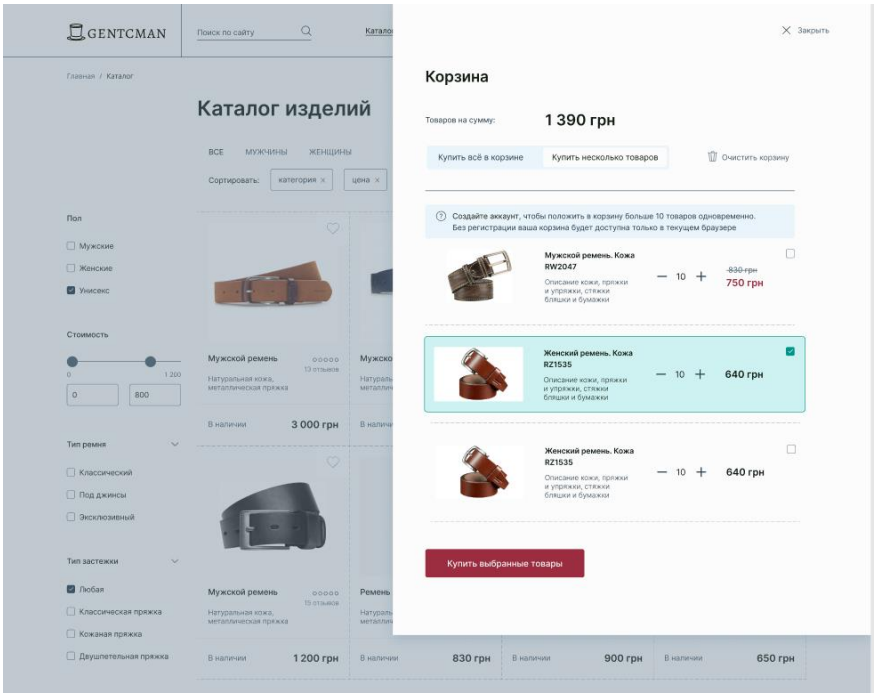

Рисунок 3.15 – Перегляд кошика

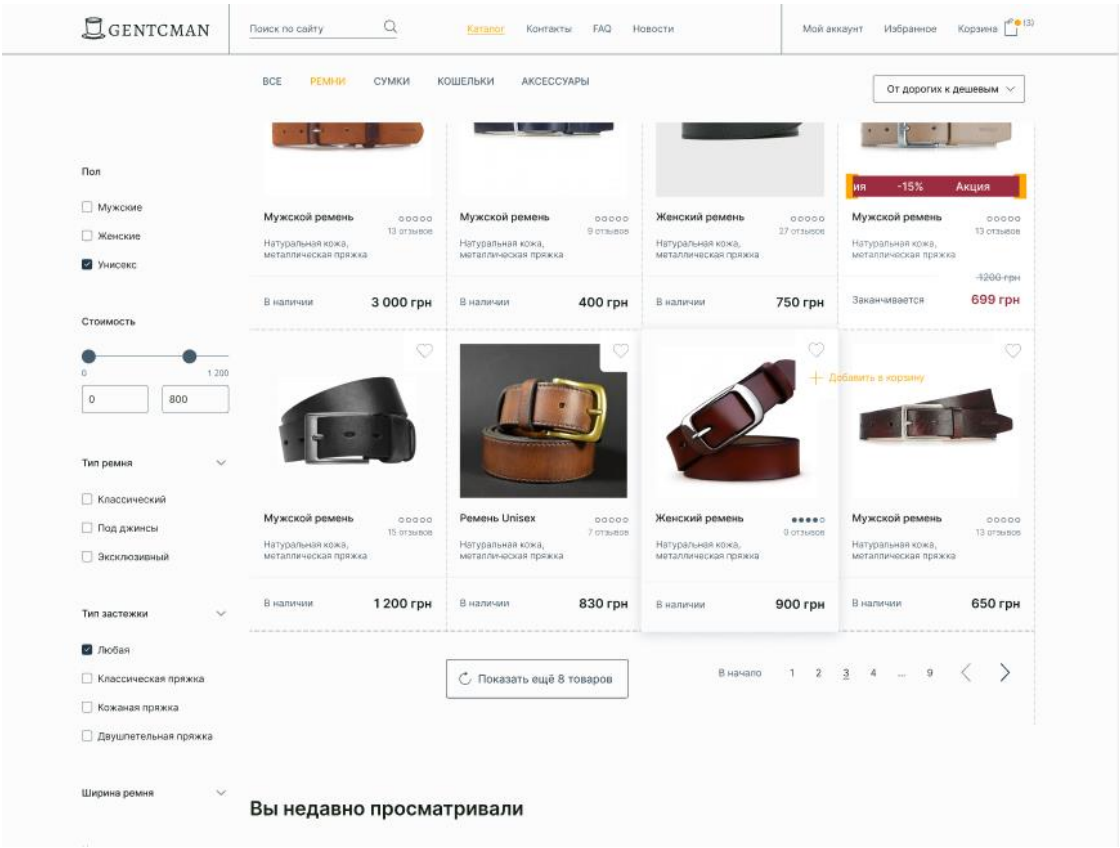

Рисунок 3.16 – Каталог товарів

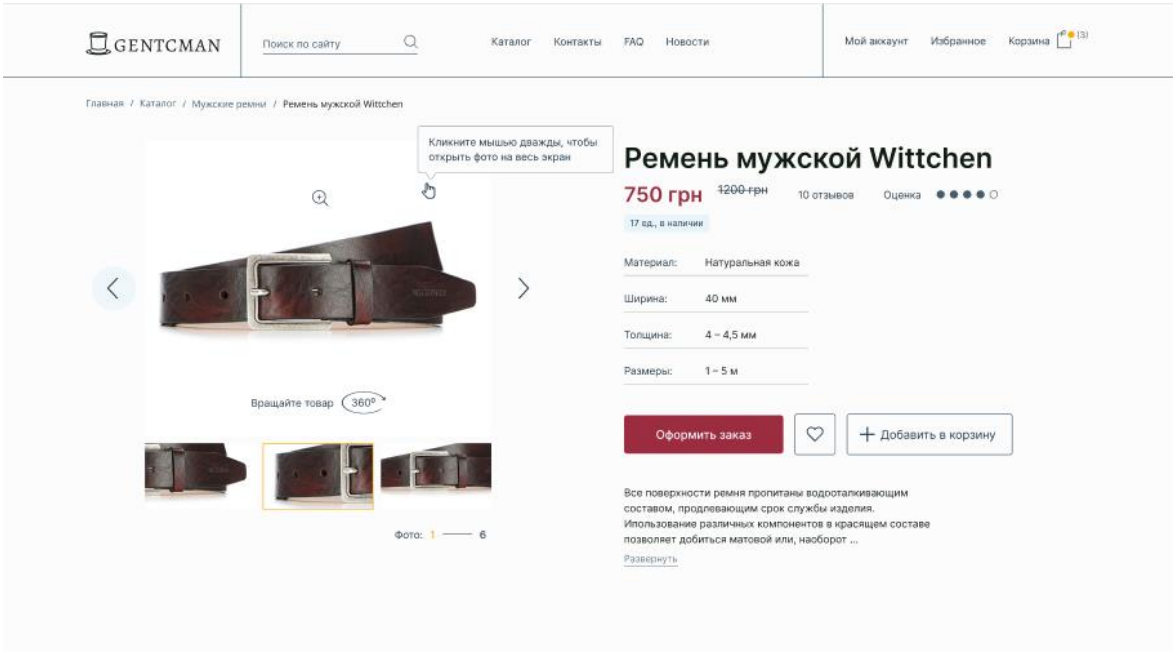

Рисунок 3.17 – Сторінка товару

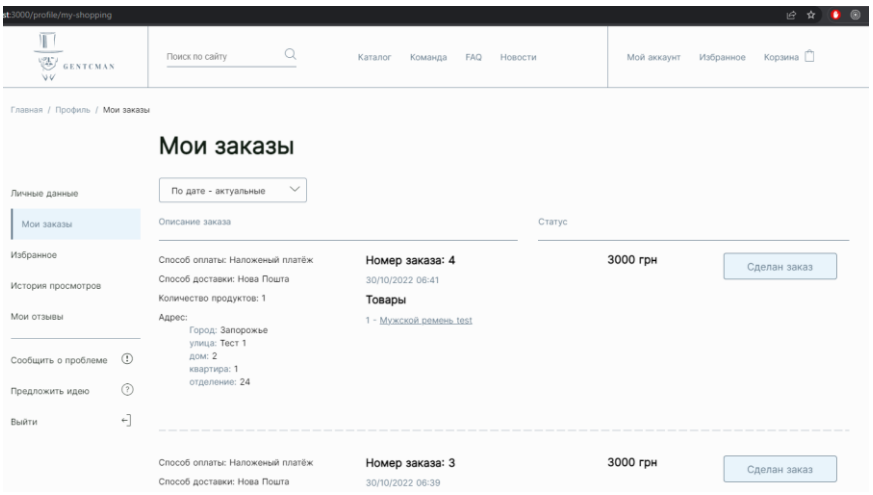

Рисунок 3.18 – Сторінка мої замовлення

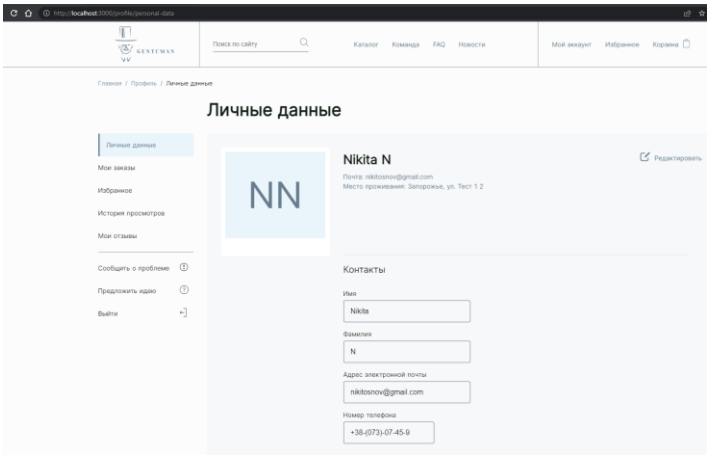

Рисунок 3.19 – Сторінка обліковий запис користувача

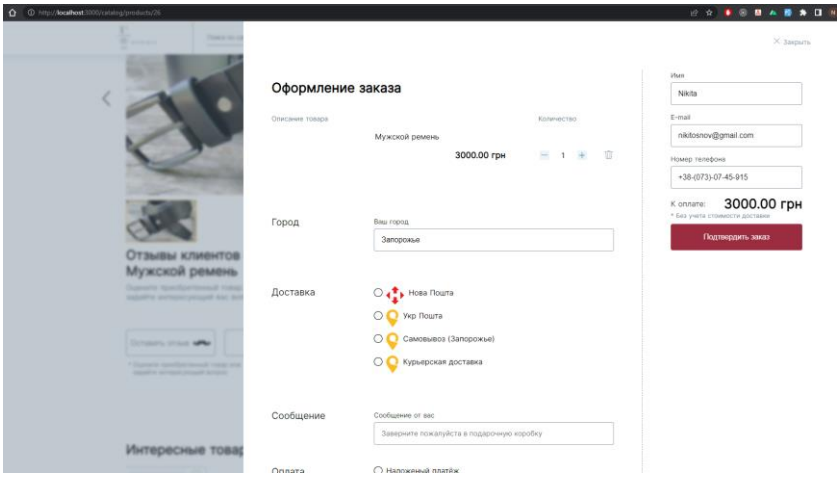

Рисунок 3.20 – Сторінка оформлення замовлення

ВИСНОВКИ

В ході роботи був розроблений веб-додаток «Інтернет магазин продаж шкіряних виробів з використанням Laravel та Nuxt.js».

У першому розділі було проаналізовано характеристику предметної області, визначені вимоги до веб-додатку, аналіз готових рішень і огляд використаних технологій.

У другому розділі було розглянуто ТЗ та продемонстровано аналіз бізнес-процесів інформаційної системи, різні діаграми, які описують розробку проекту.

У третьому розділі було розглянуто тестування АПІ, контейнеризація бекенд частини та реалізовано веб додаток, який дозволяє ознайомиться з каталогом товарів магазину, а також оформити замовлення.

На мою думку, інформаційну систему можна поліпшити в деяких місцях, а саме:

1) додати мультимовність на сайт;
2) додати можливість оплачувати товар прямо на сайті, не чекаючи дзвінка оператора;
3) розширити асортимент товарів;
4) можливість генерувати з замовлення PDF файл і можливість завантажити його зі свого облікового запису користувача;
5) реалізувати парсери для моніторингу цін на товари конкурентів, щоб можна було аналізувати дані у адмін панелі;
6) в базі даних додати індексацію даних, для збільшення продуктивності по отриманню записів.

Я вважаю, що даний проєкт можна досить непогано розвинути надалі, вийде досить цікавий комерційний продукт.

ПЕРЕЛІК ПОСИЛАНЬ

1. MySQL документація. URL: https://www.mysql.com (дата звернення: 10.09.2022).

2. Laravel документація. URL: https://laravel.com/docs (дата звернення: 12.09.2022).

3. Nuxt js документація. URL: https://ru.nuxtjs.org/docs/2.x/get-started/installation (дата звернення: 12.09.2022).

4. Vue js документація. URL: https://vuejs.org/v2/guide/ (дата звернення: 21.09.2022).

5. Патерни проектування PHP. URL: https://refactoring.guru/ru/design-patterns/php/ (дата звернення: 20.10.2022).

6. Шаблони проектування в JavaScript. URL: https://medium.com/@marina.kovalyova/java-script-design-patterns-569c627d25f9 (дата звернення: 20.10.2022).

7. Авторизація з використанням Laravel. URL: https://laravel.com/docs/8.x/authorization#introduction (дата звернення: 24.10.2022).

8. Docker документація. URL: https://www.docker.com/ (дата звернення: 02.11.2022).

9. Zandstra M. PHP Objects, Patterns, and Practice: 3rd edition. Apress, 2010. 354 p.

10. Molinaro A. SQL Cookbook: Query Solutions and Techniques for Database Developers: 1st edition. Sebastopol, 2005. 877 p.

11. Резіг Д., Бибо Б., Марас І. Секрети JavaScript ніндзя: друге видання. Київ, 2020. 377 с.

ДОДАТОК А

Приклад тестування АПІ для логіну

А.1 Фрагмент коду: тести для покриття функціоналу логіну

```php
class LoginTest extends TestCase
{
    use DatabaseTransactions;

    protected string $baseUrl = '/api/v1/client/sign-in';

    public function testSuccessfulLogin() {
        $user = User::factory()->createOne([
            'password' => bcrypt($password = 'i-love-laravel'),
        ]);

        $this->json('post', $this->baseUrl, [
            "email" => $user->email,
            "password" => $password,
        ])
            ->assertStatus(Response::HTTP_OK)
            ->assertJsonStructure(
                [
                    "success",
                    "message",
                    "data" => [
                        'logged_in_user_id',
                        'token'
                    ]
                ]
            );
    }

    public function testFailedLogin()
    {
        $payload = [
            "name" => "fake name",
            "email" => "fake@gmail.com",
            "password" => "fake_password",
        ];

        $this->json('post', $this->baseUrl, $payload)
            ->assertStatus(Response::HTTP_BAD_REQUEST)
            ->assertExactJson(
                [
                    "success" => false,
                    "error" => [
                        "details" => "Provide correct credentials",
                        "message" => "Bad Request"
                    ],
                ]
            );
    }
```

```
    public function testRequiredPasswordAndEmailFieldsForLogin() {
        $this->json('post', $this->baseUrl, [])
            ->assertStatus(Response::HTTP_UNPROCESSABLE_ENTITY)
            ->assertExactJson([
                "success" => false,
                "error" => [
                    "message" => [
                        "password" => ["The password field is required."],
                        "email" => ["The email field is required."],
                    ],
                    "details" => "Validation Error.",
                ]
            ]);
    }
```

ДОДАТОК Б

Приклад тестування АПІ для реєстрації

Б.1 Фрагмент коду: тести для покриття функціоналу реєстрації

```php
class RegistrationTest extends TestCase
{
    use DatabaseTransactions;

    protected string $baseUrl = '/api/v1/client/sign-up';

    public function testRequiredFieldsForRegistration() {
        $this->json('post', $this->baseUrl, [])
            ->assertStatus(Response::HTTP_UNPROCESSABLE_ENTITY)
            ->assertExactJson([
                "success" => false,
                "error" => [
                    "message" => [
                        "password" => ["The password field is required."],
                        "email" => ["The email field is required."],
                        "name" => ["The name field is required."],
                    ],
                    "details" => "Validation Error.",
                ]
            ]);
    }

    public function testFailedCannotCreateTheSameUserAgain() {
        $fakeUser = User::factory()->createOne();

        $response = $this->post( $this->baseUrl, [
            'name' => $fakeUser->name,
            'email' => $fakeUser->email,
            'password' => "testpassword",
        ]);
        $response->assertStatus(Response::HTTP_UNPROCESSABLE_ENTITY);
        $response->assertExactJson(
            [
                "success" => false,
                "error" => [
                    "message" => [
                        "email" => ["The email has already been taken."],
                    ],
                    "details" => "Validation Error.",
                ]
            ]
        );
    }

    public function testSuccessfulRegistration() {
        $user = User::factory()->makeOne([
            'password' => bcrypt($password = 'i-love-laravel'),
        ]);
```

```
        $response = $this->post( $this->baseUrl, [
            'name' => $user->name,
            'email' => $user->email,
            'password' => $password,
        ]);

        $response->assertStatus(Response::HTTP_CREATED);
        $response->assertExactJson(
            [
                "success" => true,
                'data' => [
                    'user_id' => $response['data']['user_id'],
                ],
                "message" => "Registered successfully"
            ]
        );

        $this->assertDatabaseHas('users', [
            'id' => $response['data']['user_id'],
        ]);
    }
```

ДОДАТОК В

Приклад тестування API для сутності продукт

В.1 Фрагмент коду: тести для покриття CRUD (створення, читання, оновлення, видалення) операцій сутності продукту

```
class ProductsTest extends TestCase
{
    use DatabaseTransactions;

    private string $productAdminApiResource = '/api/v1/admin/products';
    private string $productClientApiResource  = '/api/v1/client/products';

    public function testShouldGetAllProducts() {
        $response = $this->json('GET', $this->productClientApiResource);
        $response->assertStatus(Response::HTTP_OK);

        $response->assertJsonStructure(
            [
                "data" => [
                    "data" => [
                        "*" => [
                            'id',
                            'name',
                            'cost',
                            'meta_title',
                            'meta_description',
                            'meta_keywords',
                            'amount',
                            'rating',
                            'description',
                            'content',
                            'images_content',
                            'banner_image',
                            'published',
                            'product_status_id',
                            'created_at',
                            'updated_at',
                            'discounts'
                        ]
                    ]
                ]
            ]
        );
    }

    public function testShouldGetProductById() {
        $fakeProduct = Product::factory()->createOne();
        $response = $this->json('GET', "{$this-
>productClientApiResource}/{$fakeProduct->id}");
        $response->assertStatus(Response::HTTP_OK);

        $response->assertJsonStructure(
```

```
            [
                "data" => [
                    'id',
                    'name',
                    'cost',
                    'meta_title',
                    'meta_description',
                    'meta_keywords',
                    'amount',
                    'rating',
                    'description',
                    'content',
                    'images_content',
                    'banner_image',
                    'published',
                    'product_status_id',
                    'created_at',
                    'updated_at',
                    'discounts'
                ]
            ]
        );
    }

    public function testShouldNotGetProductById() {
        $response = $this->get("{$this->productClientApiResource}/0");
        $response->assertStatus(Response::HTTP_NOT_FOUND);
    }

    public function testShouldNotCreateProductDueToNoAuthorization() {
        $product = Product::factory()->makeOne();

        $response = $this->post($this->productAdminApiResource, $product-
>toArray());
        $response->assertStatus(Response::HTTP_UNAUTHORIZED);
    }

    public function testShouldCreateProductDueToPermission() {
        $product = Product::factory()->makeOne();
        $this->authorizeUserAs(Role::USER_NAME);
        $response = $this->post($this->productAdminApiResource, $product-
>toArray());
        $response->assertStatus(Response::HTTP_FORBIDDEN);
    }

    public function testShouldCreateProduct() {
        $product = Product::factory()->makeOne();
        $productCategory = ProductCategory::factory()->createOne();
        $product['status_id'] = $product['product_status_id'];
        $product['category_ids'] = [$productCategory->id];
        unset($product['product_status_id']);

        $this->authorizeUserAs(Role::ADMIN_NAME);
        $response = $this->post($this->productAdminApiResource, $product-
>toArray());
        $response->assertStatus(Response::HTTP_CREATED);
        $response->assertJsonStructure(
            [
                "data" => [
                    "id"
                ]
            ]
        );
        $this->assertDatabaseHas('products', [
```

```
            'id' => $response['data']['id'],
        ]);
    }

    public function testShouldUpdateProductDueToNoAuthorization() {
        $product = Product::factory()->createOne();
        $product->name = "Updated";
        $this->assertDatabaseHas('products', [
            'id' => $product->id,
        ]);
        $response = $this->put("{$this->productAdminApiResource}/{$product-
>id}", $product->toArray());
        $response->assertStatus(Response::HTTP_UNAUTHORIZED);
        $updatedProduct = Product::find($product->id);
        $this->assertStringNotContainsString($updatedProduct->name,
"Updated");
    }

    public function testShouldUpdateProductDueToPermission() {
        $product = Product::factory()->createOne();
        $product->name = "Updated";
        $this->assertDatabaseHas('products', [
            'id' => $product->id,
        ]);
        $this->authorizeUserAs(Role::USER_NAME);
        $response = $this->put("{$this->productAdminApiResource}/{$product-
>id}", $product->toArray());
        $response->assertStatus(Response::HTTP_FORBIDDEN);
        $updatedProduct = Product::find($product->id);
        $this->assertStringNotContainsString($updatedProduct->name,
"Updated");
    }

    public function testShouldUpdateProduct() {
        $product = Product::factory()->createOne();
        $productCategory = ProductCategory::factory()->createOne();
        $product['status_id'] = $product['product_status_id'];
        $product['category_ids'] = [$productCategory->id];
        $this->assertStringNotContainsString($product->name, "Updated");
        $product->name = "Updated";
        $this->authorizeUserAs(Role::ADMIN_NAME);
        $response = $this->put("{$this->productAdminApiResource}/{$product-
>id}", $product->toArray());
        $response->assertStatus(Response::HTTP_OK);
        $response->assertJsonStructure(
            [
                "data" => [
                    "id"
                ]
            ]
        );
        $updatedProduct = Product::find($product->id);
        $this->assertStringContainsString($updatedProduct->name, "Updated");
    }

    public function testShouldDeleteProductDueToNoAuthorization() {
        $product = Product::factory()->createOne();
        $this->assertDatabaseHas('products', [
            'id' => $product->id,
        ]);
        $response = $this->delete("{$this-
>productAdminApiResource}/{$product->id}");
        $response->assertStatus(Response::HTTP_UNAUTHORIZED);
        $this->assertDatabaseHas('products', [
```

```
            'id' => $product->id,
        ]);
    }

    public function testShouldDeleteProductDueToPermission() {
        $product = Product::factory()->createOne();
        $this->assertDatabaseHas('products', [
            'id' => $product->id,
        ]);
        $this->authorizeUserAs(Role::USER_NAME);
        $response = $this->delete("{$this-
>productAdminApiResource}/{$product->id}");
        $response->assertStatus(Response::HTTP_FORBIDDEN);
        $this->assertDatabaseHas('products', [
            'id' => $product->id,
        ]);
    }
```

ДОДАТОК Г

Приклад реалізації Dockerfile

Г.1 Фрагмент коду: конфігурація Dockerfile для проекту

```
FROM php:8.0-apache

# 1. Install development packages and clean up apt cache.
RUN apt-get update && apt-get install -y \
    curl \
    g++ \
    git \
    libbz2-dev \
    libfreetype6-dev \
    libicu-dev \
    libjpeg-dev \
    libmcrypt-dev \
    libpng-dev \
    libreadline-dev \
    libzip-dev \
    sudo \
    unzip \
    zip \
    nano \
 && rm -rf /var/lib/apt/lists/*

# 2. Apache configs + document root.
RUN echo "ServerName laravel-app.local" >> /etc/apache2/apache2.conf

ENV APACHE_DOCUMENT_ROOT=/var/www/html/public
RUN sed -ri -e 's!/var/www/html!${APACHE_DOCUMENT_ROOT}!g'
/etc/apache2/sites-available/*.conf
RUN sed -ri -e 's!/var/www/!${APACHE_DOCUMENT_ROOT}!g'
/etc/apache2/apache2.conf /etc/apache2/conf-available/*.conf

# 3. mod_rewrite for URL rewrite and mod_headers for .htaccess extra headers
like Access-Control-Allow-Origin-
RUN a2enmod rewrite headers

# 4. Start with base PHP config, then add extensions.
RUN mv "$PHP_INI_DIR/php.ini-development" "$PHP_INI_DIR/php.ini"

RUN docker-php-ext-install \
    bcmath \
    bz2 \
    calendar \
    iconv \
    intl \
    opcache \
    pdo_mysql \
    zip

# 5. Composer.
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer
```

```
# 6. We need a user with the same UID/GID as the host user
# so when we execute CLI commands, all the host file's permissions and
ownership remain intact.
# Otherwise commands from inside the container.sh would create root-owned
files and directories.
ARG uid
RUN useradd -G www-data,root -u $uid -d /home/devuser devuser
RUN mkdir -p /home/devuser/.composer && \
    chown -R devuser:devuser /home/devuser
```

ДОДАТОК Д

Приклад реалізації docker-compose.yml

Д.1 Фрагмент коду: конфігурація docker-compose.yml для проєкту

```yaml
version: '3.5'

services:
  laravel-app:
    build:
      context: '.'
      args:
        uid: ${UID}
    container_name: laravel-app
    ports:
      - 8000:80
    environment:
      - APACHE_RUN_USER=#1000
      - APACHE_RUN_GROUP=#1000
    volumes:
      - .:/var/www/html
    networks:
      backend:
        aliases:
          - app
  mysql-db:
    image: mysql:8.0
    container_name: mysql-db
    volumes:
      - ./run/var:/var/lib/mysql
    restart: always
    environment:
      - MYSQL_ROOT_PASSWORD=${DB_PASSWORD}
      - MYSQL_DATABASE=${DB_DATABASE}
      - MYSQL_USER=${DB_USERNAME}
      - MYSQL_PASSWORD=${DB_PASSWORD}
    networks:
      backend:
        aliases:
          - mysql-db
  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    links:
      - mysql-db
    environment:
      PMA_HOST: mysql-db
      PMA_PORT: 3306
    ports:
      - '8080:80'
    networks:
      backend:
        aliases:
          - phpmyadmin
```

ДОДАТОК Е

Приклад реалізації просунутої фільтрації продуктів

Е.1 Фрагмент коду: приклад класу, який надає можливість гнучкої фільтрації продуктів

```
class ProductFilter
{

    public static function apply(Request $filters):
\Illuminate\Contracts\Pagination\LengthAwarePaginator
    {
        $query = static::applyDecoratorsFromRequest($filters, (new Product)-
>newQuery());
        return static::getResults($query, $filters);
    }

    private static function applyDecoratorsFromRequest(Request $request,
Builder $query): Builder
    {
        foreach ($request->all() as $filterName => $value) {
            $decorator = static::createFilterDecorator($filterName);
            if (static::isValidDecorator($decorator)) {
                $query = $decorator::apply($query, $value);
            }
        }
        return $query;
    }

    /**
     * @param $name
     * @return string
     */

    private static function createFilterDecorator($name): string
    {
        $filter_param = Str::studly($name);
        return __NAMESPACE__ . "\\Filters\\" . $filter_param;
    }

    /**
     * @param $decorator
     * @return boolean
     */

    private static function isValidDecorator($decorator): bool
    {
        return class_exists($decorator);
    }

    /**
     * @param Builder $query
     * @param Request $request
```

```
     * @return \Illuminate\Contracts\Pagination\LengthAwarePaginator
     */

    private static function getResults(Builder $query, Request $request):
\Illuminate\Contracts\Pagination\LengthAwarePaginator
    {
        $take = $request->query('limit') ?? 10;
        return $query->with('productCategories.discounts', 'productStatus',
'discounts')->paginate($take);
    }
}
```

Е.2 Фрагмент коду: приклад базового інтерфейсу, який повинен реалізовувати кожен окремий фільтр класу

```
interface Filter
{
    /**
     * Apply a given search value to the builder instance.
     *
     * @param Builder $builder
     * @param mixed $value
     * @return Builder $builder
     */
    public static function apply(Builder $builder, $value);
}
```

Е.3 Фрагмент коду: приклад класу, який дозволяє здійснювати пошук за матеріалом продукту

```
class Material implements Filter
{
    /**
     * Apply a given search value to the builder instance.
     *
     * @param Builder $builder
     * @param mixed $value
     * @return Builder $builder
     */
    public static function apply(Builder $builder, $value)
    {
        $arr = explode(',', $value);
        return $builder->whereIn('content->material->value', $arr);
    }
}
```

Е.4 Фрагмент коду: приклад використання класу фільтрації продуктів у контролері

```php
class ProductController extends ApiController
{
    public function searchByQuery(Request $request): \Illuminate\Http\JsonResponse
    {
        $products = ProductFilter::apply($request);
        return $this->respondOk($products);
    }
```