НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
комп'ютерних наук

_____/Голуб Б.Л.,доц.,к.т.н./

«___»_____2025р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

На тему:

«ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ОБЛІКУ МАЙНА ПІДПРИЄМСТВА»

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ Вайганг Г.О.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
д.е.н., професор _____ Руденський Р.А.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав: _____/ Жуков О. О. /

КИЇВ - 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук
(назва кафедри)

Доц., к.т.н. _____Голуб Б.Л.
(вчене звання і ступінь) (підпис) (ініціали і прізвище)

«_____» _____2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи студенту

Жукову Олександру Олександровичу

Спеціальність 121 – «Інженерія програмного забезпечення»

1. Тема бакалаврської кваліфікаційної роботи «Програмне забезпечення обліку майна підприємства»

   Затверджена наказом ректора НУБіП від «16»_12_ 2024 р. № 2248 "С"

   2. Термін подання завершеної роботи на кафедру 25.05.2025

   3. Вихідні дані до кваліфікаційної роботи : опис програмного забезпечення

   4. Перелік питань, які/що розглядаються:

   1) Системний наліз предметної області.

   2) Інформаційне забезпечення.

   3) Програмне забезпечення.

   4) Висновки.

   Дата видачі завдання "19 " 03.2025 р

Керівник кваліфікаційної роботи:_____/ Руденський Р.А. /

Завдання прийняв до виконання: _____/ Жуков О. О. /

ЗМІСТ

ПЕРЕЛІК ВИКОРИСТАНИХ У РОБОТІ СКОРОЧЕНЬ

FTP — File Transfer Protocol — протокол передачі файлів;

HTTP — HyperText Transfer Protocol — протокол передачі гіпертексту;

HTTPS — HyperText Transfer Protocol Secure — безпечний протокол передачі гіпертексту;

MVC — Model-View-Controller — підхід до проектування архітектури Модель-Представлення-Контролер;

SQL — Structured Query Language — мова структурованих запитів;

БД — база даних;

ОС — операційна система;

ПЗ — програмне забезпечення;

ПК — персональний комп’ютер;

ПП — програмний продукт;

ІС – інформаційна система;

СУБД — система управління базою даних.

UI – user interface – інтерфейс користувача

UX – user experience – досвід користувача

ВСТУП

В еру інформаційних технологій, коли доступ до мережі інтернет є у більшості, доступ до всесвітньої павутини можна здійснити з комп'ютера, телефона, планшет та багатьох інших пристроїв – все більше і більше систем переходять в онлайн, а деякі і зовсім існують тільки в мережі.

Важливість розвитку систем обліку онлайн продемонстрували події 2022-2024 року. Повномаштабні бойові дії призвели до вимушеного зачинення фізичних приміщень підприємств, магазинів, комерційних та некомерційних компаній. Проте залишився доступ до придбання товарів та послуг через мережу інтернет. Так підприємства, що мали інтернет майданчики змогли продовжити надавати свої послуги в онлайн режимі.

В той же час набули особливої актуальності системи розміщення та обліку товарів і ресурсів без фізичного контакту з товаром, персоналом, що надає послуги за допомого мережі інтернет. Та автоматизувати процеси обліку товарів та послуг підприємства.

Веб-ресурси є чи не найдоступнішою область для широкого загалу людей. Такі застосунки неприв'язані до потужного апаратного забезпечення або операційної системи.

ІС обліку товарів дозволяє знизити витрати на оплату праці співробітників, адже ще одною вагомою перевагою інформаційних систем є автоматизація процесів отже така система буде працювати без нагляду менеджера, адміністратора тощо. Система (при виключенні можливих технічних проблем) не потребує вихідних, перерв, відпусток – вона працює щодня та цілодобово.

Такий ресурс може існувати як самостійно так і може виступати додатковим інструментом у масштабуванні системи обліку та дозволить йому функціонувати в умовах війни, пандемії. Обране середовище існування системи надають можливості для досить легкої підтримки, розширення, рефакторингу програмного забезпечення.

У завдання дипломного проекту входить аналіз, моделювання та алгоритмізація шаблону ІС обліку товарів, що дозволить надавати послуги обліку багато тематичного товару. Пошуки оптимальних рішень для більшої ефективності ресурсу та його швидкодії.

Для розробки програмного забезпечення була використана архітектура MVC (модель-представлення-контролер). Використання принципів MVC призводить до більш структурованого коду, дозволяє працювати над проектом більш спеціалізованим людям, спрощує його підтримку, робить код більш логічним і зрозумілим. Зміна в одному з компонентів мінімально впливає на інші складові. З іншого боку, це вимагає більшої продуктивності виконуючих машин, але останнім часом це не є великою проблемою - все більш складні програмістські рішення вимагають підтримки, і витрати на підтримку набагато перевищать витрати на більш потужне сучасне обладнання. MVC реалізує об'єктно-орієнтований стиль програмування, який в свою чергу є більш структурованим в порівнянні з процедурним стилем та надає більш зручні можливості для розширення програмного забезпечення. Підтримка такого коду легше так, як він слідує вельми жорстким правилам написання коду. Повторне використання класів – ООП стиль дозволяє уникнути DRY, що розшифровується як Don't Repeat Yourself (не повторюйся), тобто імовірність того, що буде створений клас який дублює інший, раніше створений, досить малі.

Для реалізації серверної частини було використано фреймворк Laravel так як цей фреймворк втілює принципи MVC, використовує PHP для

написання коду, в також для організації безпечної передачі даних, та уникання помилок при роботі з базою даних. Даний фреймворк включає в себе компоненти безпечної передачі даних. Має вбудований захист від CSRF атак та передбачає SQL ін’єкції.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

З розвитком сфери Інформаційних технологій та WEB застосунків користувач стає більш вибагливим та більш лінивим, отже розроблюваний інтернет-магазин повинен відповідати певним вимогам:

- безпечне та надійне з'єднання з базою даних, стійкість до відмов, захищеність службових даних підприємства;
- можливість виконувати основні операції «в один клік»;
- доступ до основних сторінок (каталогів, підкаталогів, деяких товарів) з головної сторінки:
- швидкий пошук на головній сторінці сервісу;
- можливість додавання товару до накладної простим рухом;
- оформлення накладної через форму з мінімальною кількістю необхідних даних (телефон);

Зі сторони адміністратора ІС обліку товарів повинні буди реалізовані наступні методи:

- додавання, видалення, редагування товару та його властивостей;
- додавання, видалення, редагування категорій продукції;
- додавання, видалення, редагування властивостей, що встановлюються для опису товару;
- перегляд історії взаємодіїз персоналом складу та постачальниками;
- редагування деякого статичного тексту;

Доступ до адміністративних каталогів повинен бути захищений, паролем.

Сервіс доступний в мережі через доменне ім’я. За допомогою користувацького інтерфейсу, який має бути зрозумілим та якомога більше простим, користувач здійснює взаємодію з елементами інтерфейсу, що дає йому змогу здійснити відгрузку товару, сформувати накладну, отримати інформацію, отримати оплату тощо. В дипломному проекті буде розглянуто ІС обліку, зберігання та видачі інструменту.

1.2 Аналіз вимог до програмної системи

1.2.1 Вимоги до технічної реалізації. Всесвітню павутину утворюють мільйони веб-серверів мережі Інтернет, розташованих по всьому світу. Веб-сервер - комп'ютерна програма, яка запускається на підключеному до мережі комп'ютері і використовує протокол HTTP для передачі даних. У найпростішому вигляді така програма отримує по мережі HTTP-запит на певний ресурс, знаходить відповідний файл на локальному жорсткому диску і відправляє його на комп'ютер як відповідь на запит. Більш складні веб-сервери здатні у відповідь на HTTP-запит динамічно генерувати документи за допомогою шаблонів і сценаріїв.

Для перегляду інформації, отриманої від веб-сервера, на клієнтському комп'ютері застосовується спеціальна програма - веб-браузер. Основна функція веб-браузера - відображення гіпертексту. Всесвітня павутина нерозривно пов'язана з поняттями гіпертексту і гіперпосилання. Велика частина інформації в Інтернеті є саме гіпертекст.

Програмне забезпечення взаємодіє з користувачем (або постачальником), адміністратором веб-сайту, сервером та базою даних. У якості бази даних використовується MySQL з веб інтерфейсом PHPMyAdmin на сервері Apache HTTP Server.

PHPMyAdmin - веб-додаток з відкритим кодом, написаний на мові PHP і представляє собою веб-інтерфейс для адміністрування СУБД MySQL. PHPMyAdmin дозволяє через браузер і не тільки здійснювати адміністрування сервера MySQL, запускати команди SQL і переглядати вміст таблиць і баз даних. Додаток користується великою популярністю у веб-розробників, так як дозволяє управляти СУБД MySQL без безпосереднього введення SQL команд.

Для розробки проекту було обрано PHPMyAdmin тому що:

- це веб-застосунок, отже доступ до нього можна отримати незалежно від апаратних та програмних властивостей пристрою;
- класичний sql синтаксис, проста взаємодія з більшістю популярних back-end фреймворків;
- реляційна база на відміну від PostgreSQL та MongoDB. В розроблюваному проекті не є суттєвим об’єктна або документо-орієнтована модель – чим простіше, тим краще;
- більшість хостингів пропонують phpMyAdmin без додаткового встановлення чи налагодження – «з коробки»;
- найчастіша модель взаємодії php + Apache HTTP Server (MongoDB nginx + nodeJS);

Сервер – будь-який Apache HTTP Server сервер з мінімальними характеристиками, що пропонуються хостинг-провайдерами на українському ринку.

Клієнт – будь-який веб браузер, що використовує користувач веб-сайту, за умови, що браузер має версію не нижче ніж на 10 версій від актуальної на момент використання інтернет-магазину.

Серверна частина повинна реалізовувати архітектуру MVC і розподіляти інформацію та функціональну частини.

Так як програмний продукт має клієнтські поля для заповнення з подальшою відправкою на сервер та взаємодією з БД, потрібно передбачити можливі слабкі місця, такі як: SQL-ін’єкції тощо.

1.2.2 Вимоги до інтерфейсу. Інтерфейс повинен бути доступним на будь- яких пристроях, та будь яких розмірах екрану. Виділяємо 3 основні розміри ширини екрану: 1024px (ПК користувачі), 768px (користувачі планшетів та нетбуків), < 425px (користувачі мобільних девайсів).

Реалізація адаптивності можлива за допомогою CSS3, а саме влаштованого в мову інтерфейсу медіа запитів та флексової моделі (подібної до флексової моделі, що використовується при розробці мобільних застосунків).

Не менш важливим є кросбраузерність, тобто однакове відображення веб-інтерфейсу на усіх популярних браузерах поточних версій (включаючи браузери на мобільних пристроях).

1.3 Огляд інформаційних джерел та існуючих рішень

В мережі інтернет існує безліч вже готових рішень поставленої проблеми, одним із таких рішень є хмарна система обліку DNTrade – станом на 2024 рік є однією з найпопулярніших систем обліку товарів України. Забезпечує якісний облік для підприємств торгівлі, складів, виробництв.

Обслуговує продажі, закіпки, склад, зберігання, рух товарів, фінансову звітність для клієнтів та постачальників.

Що собою представляє цей сервіс з точки зору користувача: хмарне сховище з великою кількістю доступних послуг з обробки та аналізу даних, згрупованих у каталоги та підкатлоги відповідно до потреб підприємства . Кожний з яких є окремим об’єктом інтерфейсу, що дає змогу перейти в потрібний каталог без попереднього переходу до батьківського каталогу.

При чому більш абстрактний батьківський (базовий) каталог – не містить послуг, а лише посилання на конкретизовані підкаталоги. Перший та другий, за рівнем вкладеності, підкаталоги будуть містити послуги. В той час як перший ще додатково буде містити посилання на підкаталоги другого рівня вкладеності (посилання на підкаталоги відносно першого рівня вкладеності, а не базового каталогу).

Каталоги з товарами в свою чергу мають фільтри, що дають змогу фільтрувати товар вибірки за умовою. Фільтри різних каталогів відрізняються, тобто їх створення є динамічним, що дає змогу більш гнучких налаштувань для користувача.

Картка товару містить інформацію про товар, характеристики, фото, вартість, а також доступну інформацію - задіяний персонал, історію операцій тощо. За необхідності по кожному товару або групі товарів може бути сформований детальний звіт, відповідно до потреб користувача.

«Запит» - приведе до оформлення даних, що знаходиться у запиті на момент натискання кнопки (під час натискання кнопки, товар на сторінці якого була натиснута кнопка буде долучено до запиту попередньо).

«Сформувати накладну» - зберігає запит у окремий файл, на основі якого формується накладна.

«Зберегти» - зберігає запит в сесію або у локальне сховище, що дозволяє зберегти інформацію на тривалий термін, даний запит не зітреться при вимкненні вікна сайту, браузера, ПК тощо.

Оформлення накладної проводиться по простій формі, яку клієнт в свою чергу заповнює. Є змога зареєструватися, авторизувати свої права доступу, що дає додаткові можливості,

такі як: он-лайн відстеження руху товару, інтерактивна допомога у заповненні форми.

1.4 Моделювання предметної області

Уніфікована мова моделювання (UML) - це графічна мова для візуалізації, уточнення, побудови та документування артефактів програмної системи. UML пропонує стандартний спосіб опису системи, включаючи концептуальні речі, такі як бізнес-процеси та системні функції, а також конкретні речі, такі як оператори мови програмування, схеми бази даних та компоненти програмного забезпечення для багаторазового використання.

В UML використовують чотири види елементів нотації:

- фігури;
- лінії;
- значки;
- написи;

1.4.1 Діаграма послідовності. Діаграма послідовності (рис. 1) відображає взаємодію об’єктів системи в динаміці. В UML взаємодія об’єктів – це обмін повідомлень між ними.

Діаграма послідовності визначає часові особливості обміну повідомлень.

Діаграму послідовності використовують для уточнення діаграми прецедентів.

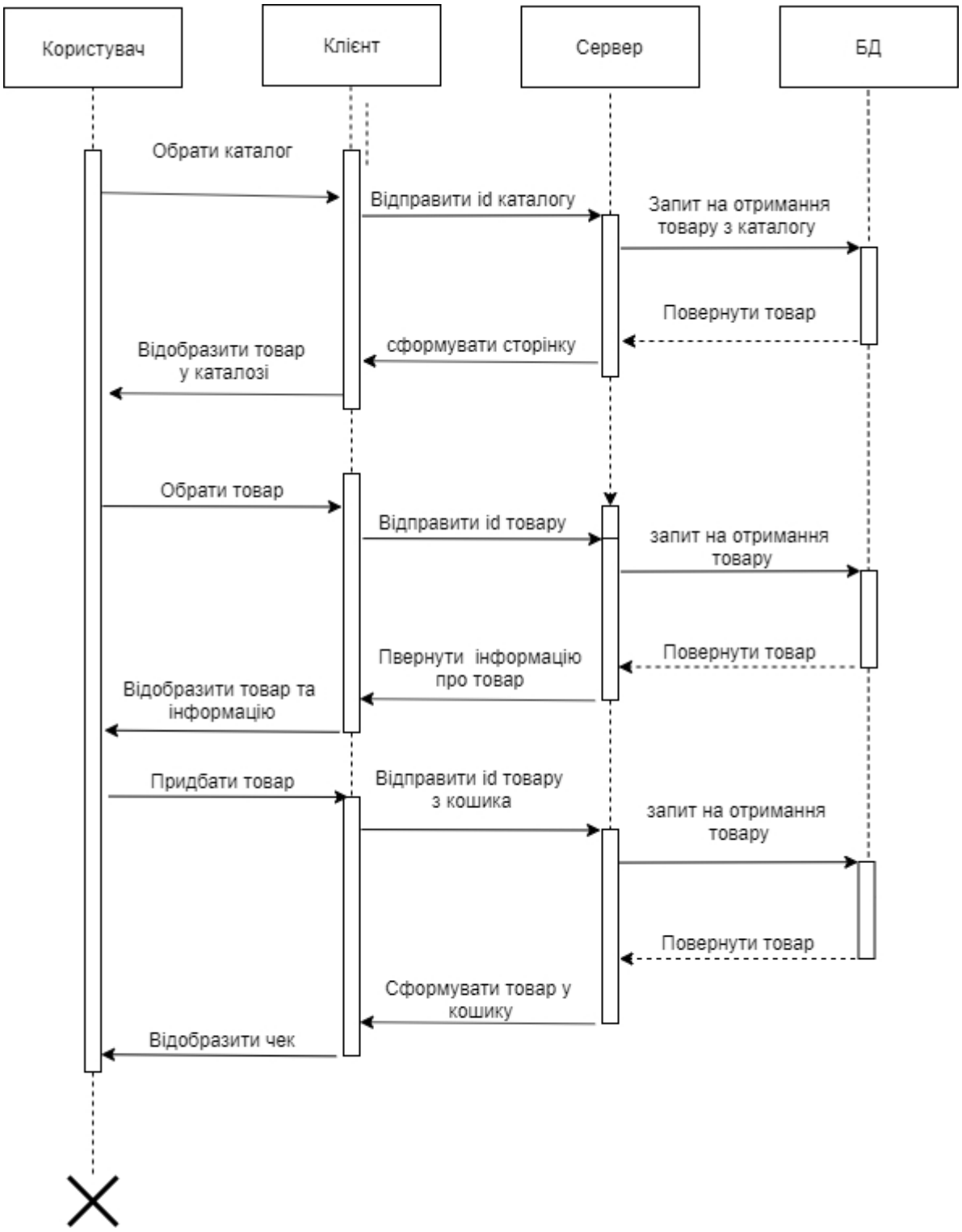


Рис. 1 Діаграма послідовності

На даній діаграмі відображена взаємодія користувача який спілкується з клієнтом (браузером). Браузер в свою чергу відправляє запити на сервер, а сервер отримує інформацію з бази даних (база даних може бути на цьому ж сервері також може бути і на віддаленому сервері).

1.4.2 Діаграма прецедентів. Діаграма прецедентів допоможе визначити границі й контекст предметної області на ранніх етапах проектування, сформулює загальні вимоги до поводження проектованої системи. Розробка концептуальної моделі системи для її наступної деталізації. Підготовка документації для взаємодії із замовником й користувачами системи.

Актор – це роль об'єкту поза системою, який прямо взаємодіє з її частиною – конкретним елементом.

Розрізняють акторів і користувачів. Користувач – фізичний об'єкт, який використовує систему. Він може грати декілька ролей і тому може моделюватися декількома акторами. Актором можуть бути різні користувачі.

Елемент – опис послідовності дій, які виконуються системою і проводять для окремого актора видимий результат. На діаграмі елемент позначається як овал.

Один актор може використовувати декілька елементів і навпаки. Кожен елемент задає певний шлях використання системи. Набір всіх елементів визначає повні функціональні можливості системи.

Між актором і елементом можливий тільки один вид відносин – асоціація, що відображає їх взаємодію. Вона може бути помічена ім'ям, ролями і потужністю.

Між акторами допустиме відношення узагальнення, що означає, що екземпляри нащадків можуть взаємодіяти з такими ж різновидами екземплярів елементів, що і екземпляр батька.

Між елементами визначені відносини узагальнення і два різновиди відношення залежності – включення і розширення.

Відношення узагальнення фіксує, що нащадок успадковує поведінку батька. Крім того, нащадок може доповнити або перевизначити поведінку

батька. Елемент, що є нащадком, може заміщати елемент, що є батьком, в будь-якому місці діаграми.

На рис. 2 відображена діаграма прецедентів, що ілюструє відношення між акторами та прцедентами. Базовий елемент явно включає поведінку іншого елементу. Елемент, що включається, ніколи не використовується самостійно.

Проілюстровано актора – клієнт та його прецеденти. Між актором та прецедентами використовується зв’язок – асоціація. Елементи видалення товару та зміна кількості товару включені у елемент зміна кошику, що в свою чергу говорить про те, що ці елементи не можуть бути викликані самостійно.

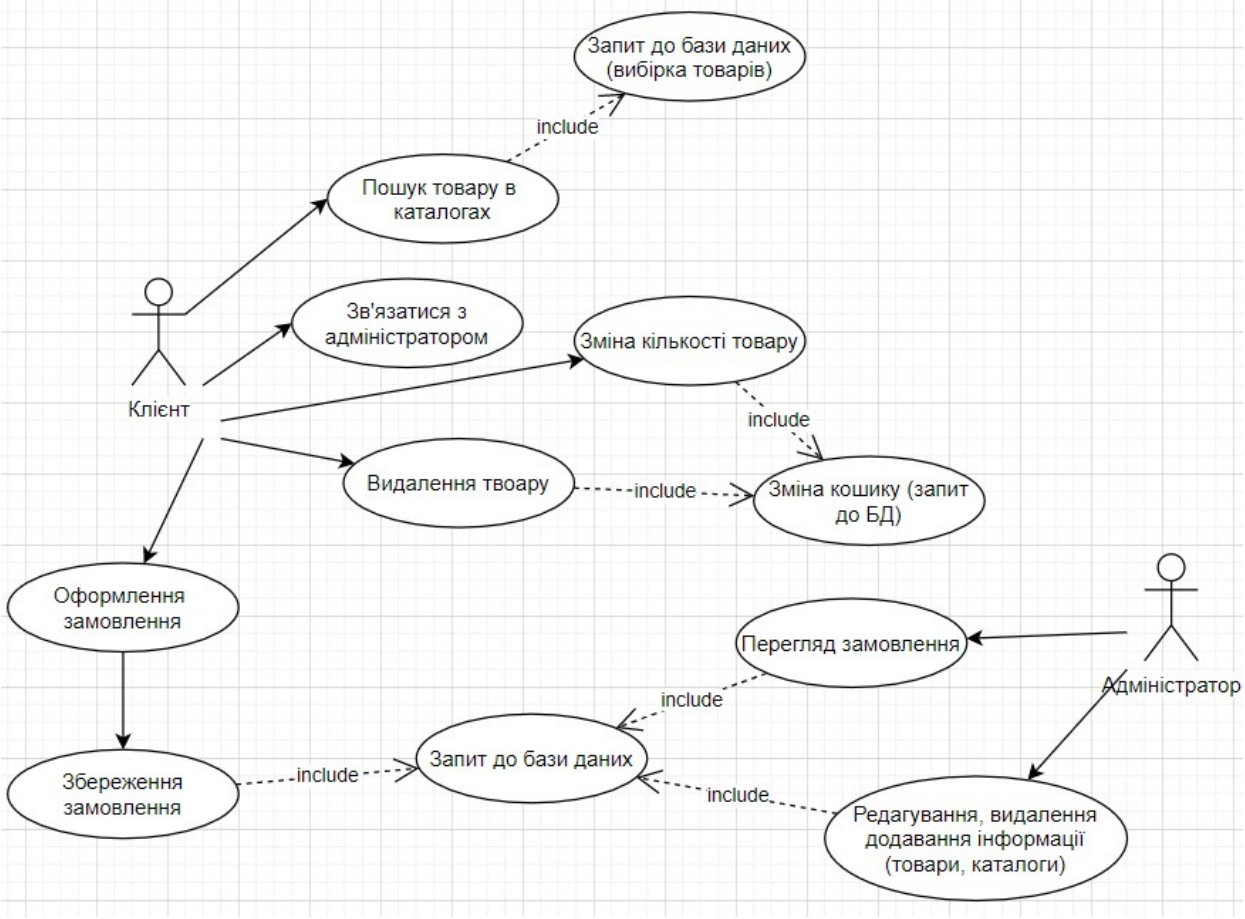


Рис. 2 Діаграма прецедентів

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних

Логічна модель даних - це версія моделі даних, яка представляє всі логічні вимоги для організації програмного забезпечення. Існують три рівні логічних моделей, які використовуються для охоплення цих вимог:

- діаграма відносин між сутностями - модель даних високого рівня, яка включає всі основні сутності та відносини. Діаграма відносин між сутностями не містить багато деталей і часто використовується на початковій фазі планування;
- базова ключова модель - модель, яка описує основні структури даних, такі як сутності, первинні ключі та атрибути вибірки;
- повно атрибутна модель - повна модель, яка включає всі необхідні сутності, атрибути, ключові групи та відносини;

Логічна модель може бути створена спільно з фізичною моделлю або незалежно від фізичної моделі. Логічні моделі також можуть бути отримані з інших моделей за допомогою майстра виведення моделей (при використанні Erwin).

Крім того, можна визначити об'єкти моделі в логічній моделі лише як логічну, а у фізичній - лише як фізичну. Ці параметри дозволяють повністю нормалізувати логічну модель і денормалізувати відповідну фізичну модель.

На рис. 3 продемонстрована логічна модель даних.

Розглянемо відносини між сутностями:

Відносини між сутностями «Товар» та «Замовлення» - багато до багатьох. Одне замовлення може містити багато товарів та один товар може належати багатьом замовленням.

Відносини між сутністю «Товар» та «Властивості» - багато до багатьох. Один товар може мати декілька властивостей та одна властивість можуть належати багатьом товарам.

Відносини між сутностями «Товар» та «Категорія» - один до багатьох. У одного товару може бути одна категорія, але до однієї категорії може належати декілька товарів.

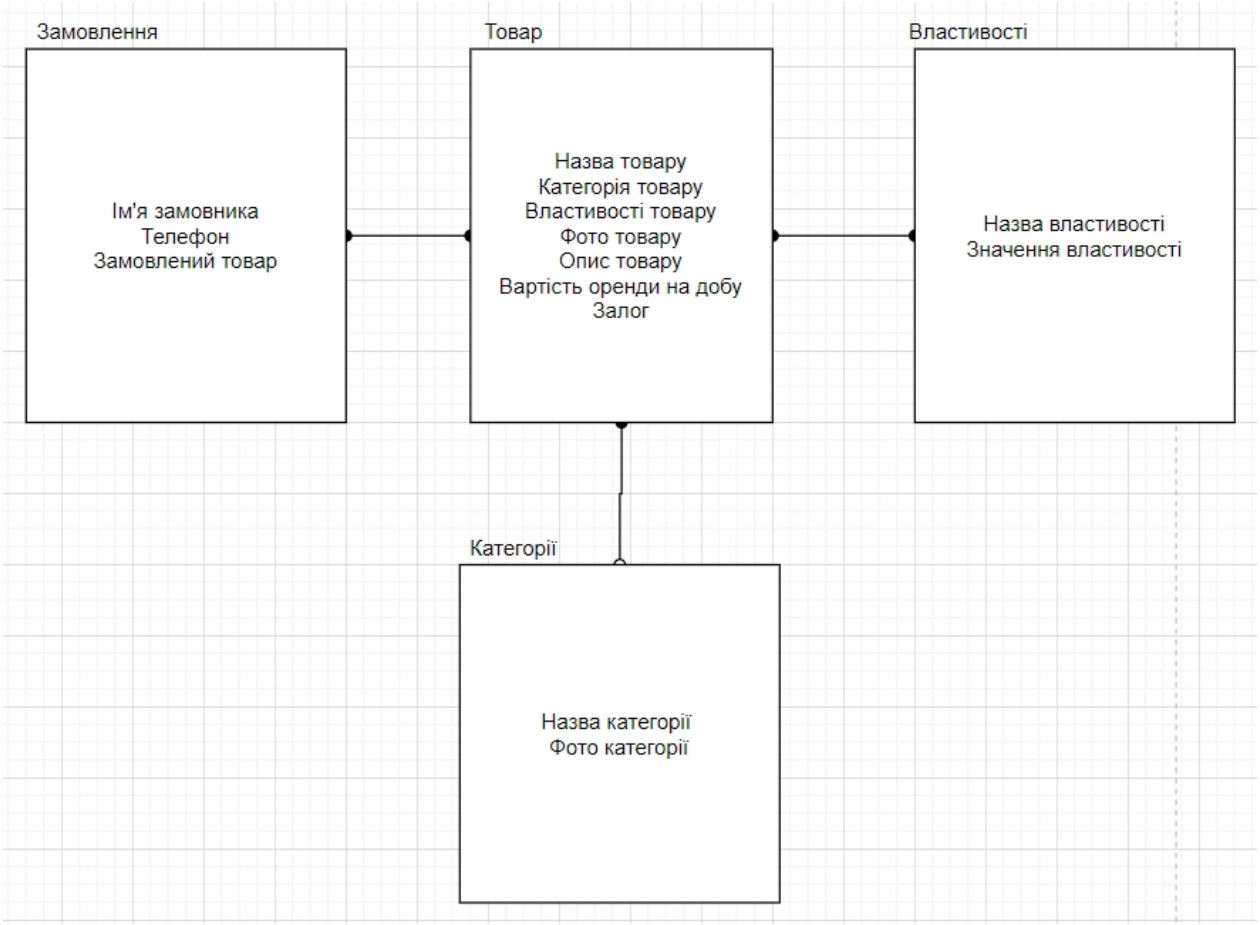


Рис. 3 Логічна модель даних

2.2 Вибір системи управління інформаційною системою

PHPMyAdmin - веб-додаток з відкритим кодом, написаний на мові PHP і представляє собою веб-інтерфейс для адміністрування СУБД MySQL.

PHPMyAdmin дозволяє через браузер і не тільки здійснювати адміністрування сервера MySQL, запускати команди SQL і переглядати вміст таблиць і баз даних. Додаток користується великою популярністю у веб-розробників, так як дозволяє управляти СУБД MySQL без безпосереднього введення SQL команд.

MongoDB - система управління базами даних (СКБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць. Класифікована як NoSQL, використовує JSON-подібні документи і схему бази даних. Написана на мові C ++. Є документо-орієнтованої системою.

PostgreSQL - об'єктно-реляційна система управління базами даних (СКБД). Існує в реалізаціях для безлічі UNIX-подібних платформ, включаючи AIX, різні BSD-системи, HP-UX, IRIX, Linux, macOS, Solaris / OpenSolaris, Tru64, QNX, а також для Microsoft Windows.

Для використання у проекті було обрано СУБД PHPMyAdmin через ряд причин описаних в розділі 1.2.1.

2.3 Створення інформаційної бази

Щоб створити базу даних можна використати один з трьох інструментів: SQL, за допомогою інтерфейсу або через міграції. Розглянемо кожен з них.

2.2.1 Створення інформаційної бази за допомогою SQL. SQL – декларативна мова програмування, застосовуэться для створення, модифікації та управління даними в реляційній базі даних, керованої відповідною системою управління базами даних.

Для створення бази даних використовується команда CREATE DATABASE. Вона має наступний синтаксис: CREATE DATABASE [IF NOT EXISTS] ім'я_бази_даних.

Створення таблиці категорій методами SQL запитів (рис. 4 та рис. 5):

```
CREATE TABLE `categories` (
  `id` bigint(20) UNSIGNED NOT NULL,
  `name` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `code` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `description` text COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `iamge` text COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL
)
```

Рис. 4 Створення таблиці категорій методами SQL

Створення первинного ключа:

```
ALTER TABLE `categories`
  ADD PRIMARY KEY (`id`);
ALTER TABLE `categories`
  MODIFY `id` bigint(20) UNSIGNED NOT NULL AUTO_INCREMENT, AUTO_INCR
EMENT=6;
```

Рис. 5 Створення первинного ключа через запити

Таким чином буде створена база даних з заданим іменем та таблиця категорій з відповідними рядками та створений первинний ключ для рядка id. Весь код створення бази даних для проекту наведено в додатку А.

2.2.2 Створення інформаційної бази за допомогою веб-інтерфейсу PHPMyAdmin. PHPMyAdmin надає простий та зручний інструментарій для створення бази даних через веб-інтерфейс. Достатньо зайти у інтерфейс у верхньому лівому кутку натиснути кнопку «Створити БД». Задати ім’я бази даних та тип кодування (якщо потрібно). Далі слід натиснути кнопку «вперед» - готово, база даних з заданим ім’ям створена.

Щоб додати таблицю слід обрати створену базу даних в у списку зліва. В відкритому меню зліва обрати пункт «Нова». Задати необхідні налаштування, імена та типи даних для рядків, задати первинний ключ та натиснути «Вперед» - таблиця створена

2.2.3 Створення інформаційної бази за допомогою міграцій. Міграції - щось на зразок системи контролю версій для бази даних. Вони дозволяють команді змінювати структуру БД, в той же час залишаючись в курсі змін інших учасників. Саме цим методом було створено більшість таблиць бази даних для проекту.

Фасад Laravel Schema забезпечує підтримку створення та зміни таблиць в незалежності від СУБД з числа тих, що підтримуються в Laravel.

Для створення нової міграції використовується Artisan-команда make: migration:

php artisan make: migration create_users_table

Міграція буде поміщена в папку database/migrations і буде містити мітку часу, яка дозволяє фреймворку визначати порядок застосування міграцій.

Можна також використовувати параметри --table і --create для вказівки імені таблиці і того факту, що міграція буде створювати нову таблицю (а не змінювати існуючу). Ці параметри заздалегідь створюють посилання на відповідну таблицю зазначену в створюваному файлі-заглушці міграції.

Клас міграцій містить два методи: up() і down(). Метод up() використовується для додавання нових таблиць, стовпців або індексів в базу даних, а метод down() просто скасовує операції, виконані методом up().

Для запуску всіх необхідних міграцій використовується Artisan-команда migrate: php artisan migrate.

Для скасування змін, зроблених останньою міграцією, використовується команда rollback. Ця команда скасує результат останньої «партії» міграцій, яка може включати кілька файлів міграцій: php artisan migrate:rollback.

На рис. 6 можна побачити міграцію, що створюють таблиці в базі даних. Назва кожної міграції є смисловою: дата та час створення міграції. Функція, що виконує міграція (створення таблиці, редагування таблиці, створення сполучної таблиці тощо).

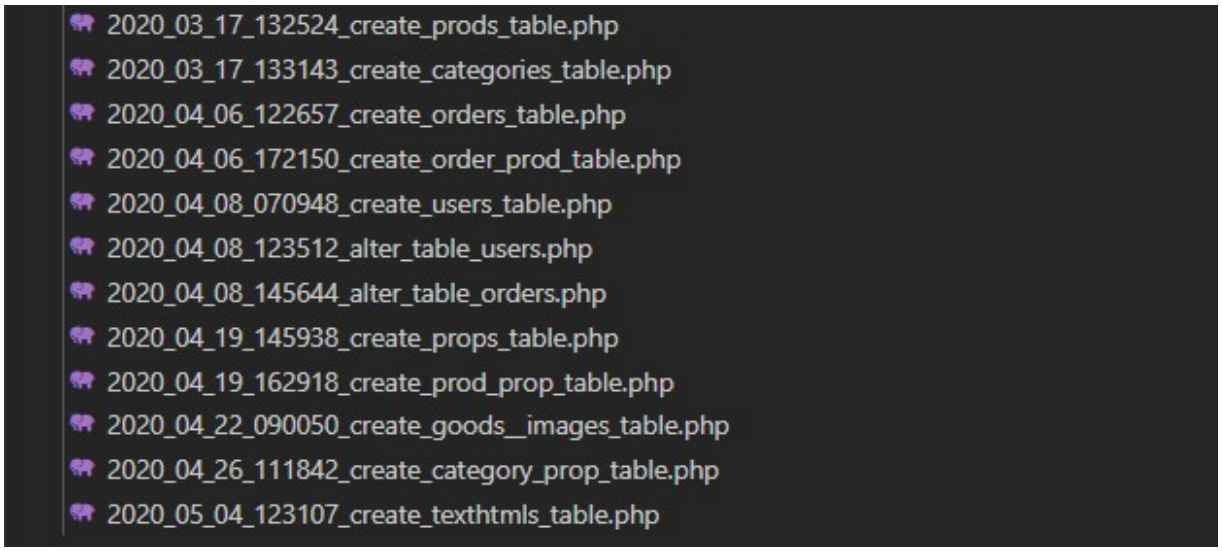


Рис. 6 Міграції виконані у проекті

На рис. 7 приведений приклад коду міграції. Метод up(), що виконається під час міграції створить таблицю prods з наступними полям:

- Id – унікальний ідентифікатор товару в таблиці, автоінкермент;
- Name – назва товару, строкове поле;
- Category_id – номер категорії до якої належить товар, число;
- Price – вартість товару, число з плаваючою крапкою за замовчуванням буде встановлено нуль;
- Articul – артикул товару, текстове поле;
- Description – опис товару, текстове поле;
- Image – основне фото товару, за замовчуванням нуль, зберігаю шлях до картинки на сервері, текстове поле;

- Timestamps - створює поля часу створення товару та часу обновлення запису

```php
public function up()
{
    Schema::create('prods', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->integer('category_id');
        $table->double('price')->default(0);
        $table->string('articul');
        $table->text('description')->nullable();
        $table->text('iamge')->nullable();
        $table->timestamps();
    });
}
```

Рис. 7 Міграція таблиці товарів

На рис. 8 розглянуто метод up() міграції для таблиці категорій:

- Id – унікальний ідентифікатор категорії в таблиці, число, автоінкермент;
- Name – назва категорії, строкове поле;
- Code – транслітерована назва категорії, строкове поле;
- Description – опис категорії, текстове поле за замовчуванням null;
- Image – основне фото категорії, зберігає шлях до картинки на сервері, текстове поле за замовчуванням null;
- Timestamps - створює поля часу створення категорії та часу обновлення запису;

```php
public function up()
{
    Schema::create('categories', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('code');
        $table->text('description')->nullable();
        $table->text('iamge')->nullable();
        $table->timestamps();
    });
}
```

Рис. 8 Міграція категорії

На рис. 9 розглянуто метод up() міграції для таблиці замовлення (orders):

- Id – унікальний ідентифікатор замовлення в таблиці, число, автоінкермент;
- Status – статус замовлення, зберігає булеве значення 0 або 1, за замовчуванням 0;
- Name – ім’я користувача, що оформлює замовлення за замовчуванням 0, строкове поле;
- Phone – телефон користувача, що оформлю замовлення, за замовчуванням 0, строкове поле;
- Timestamps - створює поля часу створення замовлення та часу обновлення запису;

```php
public function up()
{
    Schema::create('orders', function (Blueprint $table) {
        $table->id();
        $table->tinyInteger('status')->default(0);
        $table->string('name')->nullable();
        $table->string('phone')->nullable();
        $table->timestamps();
    });
}
```

Рис. 9 Міграція таблиці замовлень

Міграція на рис. 10 показує процес додавання нового поля user_id, що відповідає за ідентифікацію користувача який робить запит або додає товар у накладну, в існуючу таблицю Orders.

```php
public function up()
{
    Schema::table('orders', function (Blueprint $table) {
        $table->integer('user_id')->default(0);
    });
}
```

Рис. 10 Міграція додавання нового поля в існуючу таблицю Orders

Зведені або сполучні таблиці використовуються при стосунках між сутностями багатьох до багатьох (Many-to-many). Назва сполучної таблиці повинна містити назви таблиць в однині, що вона поєднує та в алфавітному порядку. Так фреймворк за замовчуванням буде очікувати ідентифікатори таблиць, що лягли в назву сполучної таблиці. На рис. 11 описаний код міграції, що з'єднує таблиці товарів та замовлення, назва таблиці order_prods (в алфавітному порядку). Одне замовлення може мати декілька товарів та один товар може входити до декількох замовлень. Зведена таблиця через унікальні ідентифікатори поєднує замовлення з товаром, що воно включає. Також дана

таблиця зберігає значення кількості (count). Значення count змінюється в залежності від кількості входжень одного екземпляру товару в одне замовлення. Структура міграції з рис. 11 наступна:

- Id – унікальний ідентифікатор запису зведеної таблиці, число, автоінкермент;
- Odred_id – унікальний ідентифікатор замовлення, число;
- Prods_id – унікальний ідентифікатор товару, число;
- Count – кількість входжень одного товару до замовлення, число;
- Timestamps - створює поля часу створення запису та часу обновлення запису;

```
public function up()
{
    Schema::create('order_prods', function (Blueprint $table) {
        $table->id();
        $table->integer('order_id');
        $table->integer('prods_id');
        $table->integer('count')->default(1);
        $table->timestamps();
    });
}
```

Рис. 11 Міграція зведеної таблиці замовлення та товару

Таблиця користувачів на рис. 12 має наступні поля:

- Id – унікальний ідентифікатор користувача, число, автоінкермент;
- Name – ім’я користувача, строкове значення;
- Email – електронна пошта користувача виступає в ролі логіна при авторизації, строкове унікальне поле;
- Password – пароль для авторизації користувача, строкове поле;
- Remebertoken – csrf токен користувача під час авторизації, фігурує в передачі даних на сайті;

- Timestamps - створює поля часу створення користувача та часу обновлення запису;

```php
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('email')->unique();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password');
        $table->rememberToken();
        $table->timestamps();
    });
}
```

Рис. 12 Міграція таблиці користувачів

Міграція на рис. 13 показує процес додавання нового поля is_admin, що відповідає за ідентифікацію адміністратора в існуючу таблицю Users.

```php
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->tinyInteger('is_admin')->default(0);
    });
}
```

Рис. 13 Міграція додавання нового поля в існуючу таблицю Users

Міграція на рис. 14 створить таблицю Props з наступними полями:

- Id – унікальний ідентифікатор властивості, число, автоінкермент;
- Code – транслітерована назва властивості, строкове поле;
- Name – назва властивості, строкове поле;
- Unit – одиниці виміру властивості, якщо таке потребується, строкове поле за замовчуванням NULL;

- Timestamps - створює поля часу створення властивості та часу обновлення запису;

```php
public function up()
{
    Schema::create('props', function (Blueprint $table) {
        $table->id();
        $table->string('code');
        $table->string('name');
        $table->string('unit')->nullable();
        $table->timestamps();
    });
}
```

Рис. 14 Міграція для таблиці властивостей товару

Дана сполучна таблиця (рис. 15) пов’язує товари з властивостями, окрім стандартних, для таблиці даного типу, полів з ідентифікаторами: prod_id – ідентифікатор товару, prop_id – ідентифікатор властивості, таблиця також містить поле count – фактично це значення властивості. В подальшому було вирішено перейменувати рядок у unit та використовувати строковий тип даних оскільки значення властивості може бути не обов’язково числове.

```php
public function up()
{
    Schema::create('prod_prop', function (Blueprint $table) {
        $table->id();
        $table->integer('prod_id');
        $table->integer('prop_id');
        $table->integer('count')->default(1);
        $table->timestamps();
    });
}
```

Рис. 15 Міграція сполучної таблиці товарів з властивостями

Міграція на рис. 16 створює таблицю, що відповідає відношенню «Один до багатьох». Так один товар може мати багато додаткових фото. Поля таблиці, що описує міграція:

- Id – унікальний ідентифікатор фото, число, автоінкремент;
- Prods_id – ідентифікатор товару до якого відноситься фото, число;
- Iamge – шлях та назва до фото на сервері, строкове поле;
- Timestamps - створює поля часу створення фото та часу редагування запису;

```php
public function up()
{
    Schema::create('goods__images', function (Blueprint $table) {
        $table->id();
        $table->integer('prods_id');
        $table->text('iamge');
        $table->timestamps();
    });
}
```

Рис. 16 Міграція таблиці додаткових фото товарів

Сполучна таблиця на рис. 17 пов’язує властивості з категоріями. Такий зв’язок націлений спростити роботу адміністратора при редагуванні властивостей та створенні нових товарів. При великій кількості властивостей стає не створювати товари, скільки властивості будуть виведені на екран навіть якщо вони ніяк не пов’язані з даним товаром та з’являється шанс дублювання товару. Щоб уникнути описаних проблем було вирішено розділяти властивості на групи за категорією. Так ми позбуваємося проблеми скупчення властивостей при створенні товару, оскільки будуть виводитися лише ті властивості, що відповідають обраній категорії. Та зменшуємо шанс дублювання властивостей, оскільки стає можливим додати фільтр за категорією для більш чіткого та структурованого огляду існуючих категорій.

```php
public function up()
{
    Schema::create('category_prop', function (Blueprint $table) {
        $table->id();
        $table->integer('category_id');
        $table->integer('prop_id');
        $table->timestamps();
    });
}
```

Рис. 17 Сполучна таблиця категорій та властивостей

Міграція на рис. 18 творю таблицю texthmls в базі даних яка відповідає за статичний контент сайту. Банери, сторінки «Контакти», «Доставка», банери та контент на головній сторінці, теги H1, meta-теги ті інше. Таблиця містить наступні поля:

- Id – унікальний ідентифікатор запису, число, автоінкермент;
- Page – описує сторінку на якій відображається контент, строкове значення;
- Descr – примітка, що описує де саме відображається контент, строкове поле;
- Content – контент який буде відображатися, текстове поле;
- Title – можливий заголовок якщо він потрібен, текстове поле;
- Page_code – транслітерована назва сторінки, строкове значення;
- Locate_code – транслітерована назва поля descry, строкове значення;
- Timestamps - створює поля часу створення запису та часу редагування запису;

```php
public function up()
{
    Schema::create('texthtmls', function (Blueprint $table) {
        $table->id();
        $table->string('page');
        $table->string('descr');
        $table->text('content');
        $table->text('title');
        $table->string('page_code');
        $table->string('loacte_code');
        $table->timestamps();
    });
}
```

Рис. 18 Міграція для таблиці контенту сторінок

Було розглянуто три методи для створення інформаційної бази даних для проекту з використанням різних інструментів. Можна виділити метод міграцій як найбільш пристосований для роботи у команді та великих проектів через те, що він дає змогу слідкувати за змінами, що були внесені у базу даних, структуру таблиць.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Інструменти для створення проекту

Для створення проекту було вирішено використовувати фреймворк Laravel. Щоб розробляти застосунки на даному фреймворку використовуються наступні компоненти:

PHP - розшифровується як "PHP: Hypertext Preprocessor" - «PHP: препроцесор Гіпертексту», є поширеним інтерпретується мовою загального призначення з відкритим вихідним кодом. PHP створювався спеціально для ведення web-розробок і код на ньому може впроваджуватися безпосередньо в HTML-код. Синтаксис мови бере початок з C, Java і Perl, і є легким для вивчення. Основною метою PHP є надання web-розробникам можливості швидкого створення динамічно генерованих web-сторінок, однак сфера застосування PHP не обмежується тільки цим.

Composer - менеджер залежностей для PHP.

Artisan - інтерфейс командного рядка, який поставляється з Laravel. Він містить набір корисних команд, які допомагають при розробці програми. Щоб переглянути список доступних команд використовується команда list:.

Перш за все встановлюється PHP (в даному проекті, використовувалась версія PHP 7.4).

Наступним кроком э встановлення Composer. Під час встановлення слід вказати шлях до директорії в якій знаходиться PHP.

Останній крок є розгортання проекту за допомогою командної строки.

Ці дії повинні бути виконані саме в такому порядку, як вони описані вище. Результатом, буде розгорнуто Laravel проект з усіма необхідними для його роботи модулями та залежностями.

Для налаштування проекту та подальшої розробки слід відредагувати створений за замовчуванням файл .env. А саме наступні параметри:

- APP_URL – домен або локальна адреса сервера;
- DB_HOST – сервер бази даних;
- DB_PORT – порт бази даних ;
- DB_DATABASE – ім’я бази даних, що використовується у проекті;
- DB_USERNAME – користувач цієї бази даних;
- DB_PASSWORD – пароль користувача;

3.2 Алгоритмізація та програмування програмних модулів

Структуру програмного забезпечення можна розглянути за допомогою діаграми компонентів, що входять в це програмне забезпечення. Діаграма компонент відображає залежності між компонентами програмного забезпечення, включаючи компоненти вихідних кодів, бінарні компоненти, та компоненти, що можуть виконуватись.

На рис. 19 відображена діаграма компонентів проекту. З діаграми наглядно видно як було реалізовано архітектуру MVC. Три основних компоненти: model.php, view.php та controller.php реалізують цю архітектуру за принципом розділення даних застосунку, користувацького інтерфейсу та керуючої логіки.

Модель (Model) надає дані і реагує на команди контролера, змінюючи свій стан.

Представлення (View) відповідає за відображення даних моделі користувачеві, реагуючи на зміни моделі.

Контролер (Controller) інтерпретує дії користувача, сповіщаючи модель про необхідність змін.

Розглянемо кожен з цих та додаткових компонентів окремо.

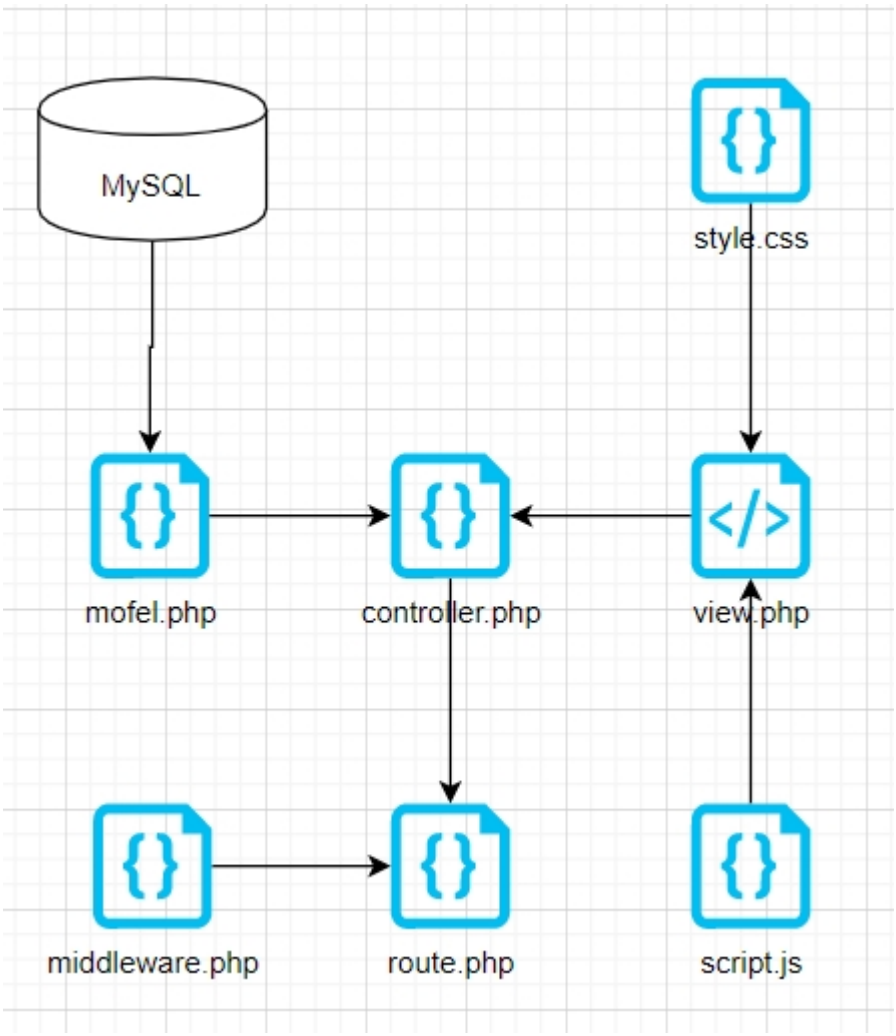


Рис. 19 Діаграма компонентів

3.2.1 Маршрутизація (роутинг) та контролери. Для обробки запиту за допомогою маршрутизації існують щонайменше два методи: обробляти запит через замикаючу функцію безпосередньо у функції, що оброблює маршрут та обробляти запит через контролер.

Замість того, щоб визначати всю логіку обробки запиту в замикаючих функціях маршрутів та писати більш структурований та розділений код, можна організувати цю обробку за допомогою контролерів. Контролери можуть групувати пов'язані однією або подібною логікою запити в один клас.

Контролери в PHP являють собою звичайні класи. Для їх створення в проекті Laravel краще всього застосовувати Artisan.

Команда: php artisan make:controller <i’мя контролера> створює новий контролер за замовчуванням в директорію app/Http/Controllers.

Зауважимо, що контролер розширює базовий клас контролера, включений до Laravel. Базовий клас надає кілька зручних методів, таких як метод middleware програмного забезпечення, який може використовуватися для приєднання проміжного програмного забезпечення до дій контролера.

Базовий клас контролера було модифіковано додатковим методом для транслітерації (рис. 20). Метод було додано до базового контролера щоб уникнути дублювання, оскільки даний метод використовують більшість адміністративних контролерів. Так відпадає потреба дублювати функцію в кожний новий створений контролер.

Таким чином було створено MainController який у подальшому буде застосовуватися для маршрутизації сторінок сайту.

```php
public function generate_chpu($a)
{
    $converter = array(…
    );
    $a = strtr($a, $converter);
    $a = strtolower($a);
    $a = preg_replace('~[^-a-z0-9_]+~u', '-', $a);
    $a = trim($a, "-");
    return $a;
}
```

Рис. 20 Метод транслітерації у базовому контролері

3.2.2 Маршрутизація. Більшість маршрутів розробленого програмного забезпечення буде визначено у файлі app/route.php. Найпростіші маршрути Laravel складаються з URI та зворотного виклику функції закриття. Також в маршрут може бути переданий параметр, що дозволяє динамічно генерувати вміст сторінок.

Розглянемо сторінки Категорія та Картка товару. Такі сторінки можуть бути задані наступними маршрутами:

Route::get('/{category}', 'MainController@category')->name('category');

Route::get('/{category}/{product}', 'MainController@prod')->name('prod');

Route – клас. Get – метод класу, який вказує яким методом потрібно передавати дані. Метод get приймає два аргументи. Перший аргумент методу – URI, при переході ня який буде викликана функція замикання або метод контролера. URI мможе бути як константою так і динамічним параметром. В даному прикладі URI динамічний параметр. Такий параметр задається в фігурних дужках. Другий параметр – функція замикання, метод класу. Метод класу name – присвою ім’я (аліас) для біль зручного використання маршрутів в коді.

Таким чином, в якості параметра передається унікальне значення із бази даних окремого об’єкта і в подальшому обробляються у контролері. Метод контролеру яким потрібно обробити запит вказується через символ «@» (собачка).

Метод category в MainController, що обробляє маршрут category:

```
public function category($code)
{
    $categories = Category::get();
    $category = Category::where('code', $code)->first();
    return view('category', compact('category', 'categories'));
}
```

Цей метод приймає один параметр – code. Code – генерується при створенні категорії. Це поле являється унікальним та генерується з назви категорії замінюючи кириличні букви на латинські аналоги (транслітерація).

Метод compact - Створює масив, що містить назви змінних і їх значення.

Функція category обробляє запит за наступною логікою: в змінну category передається об’єкт класу Category з значенням code, що відповідає переданому параметру.

По закінченню, функція повертає представлення category та за допомогою методу compact передає необхідні змінні та їх значення у представлення.

Метод prod в MainController, що обрублю маршрут category:

```
public function prod($category, $prod)
{
```

```
        $prod = Prods::where('code', $prod)->first();
        $categories = Category::get();
        return view('prod', compact('prod', 'categories'));
    }
```

Цей метод приймає два параметри оскільки відповідний до цього методу маршрут будується з двох параметрів – category та code. Code – генерується при створенні товару транслітерацією кириличних символів в латинські. Це поле являється унікальним.

Category відповідає категорії до якої відноситься товар.

Функція prod обробляє запит за наступною логікою: в змінну category передається об’єкт класу Category. В змінну prod передається товар відповідно за параметром code.

По закінченню, функція повертає представлення prod та за допомогою методу compact передає необхідні змінні (prod, categories) та їх значення у представлення.

3.2.3 Посереднє програмне забезпечення (Middleware). Посереднє програмне забезпечення (middleware) забезпечує зручний механізм фільтрації HTTP-запитів, що надходять у вашу програму. Наприклад, Laravel викликає проміжне програмне забезпечення, яке підтверджує автентичність користувача вашої програми. Якщо користувач не має автентифікації, проміжне програмне забезпечення перенаправить користувача на екран входу. Однак якщо користувач має автентифікацію, проміжне програмне забезпечення дозволить запиту продовжувати роботу в додатку.

Додаткове проміжне програмне забезпечення можна написати для виконання різноманітних завдань, крім автентифікації. Програмне забезпечення CORS може нести відповідальність за додавання належних

заголовків до всіх відповідей, які виходять із вашої програми. Посереднє програмне забезпечення для реєстрації може записувати всі вхідні запити до логу.

В пакеті Laravel є кілька проміжних програм, включаючи проміжне програмне забезпечення для аутентифікації та захисту CSRF. Усі ці проміжні програми знаходяться в каталозі app / Http / Middleware.

Окрім вже включеного у проект проміжного програмного забезпечення було створене додаткове для передбачення несанкціонованого доступу до адміністративної панелі іншим користувачем.

```php
public function handle($request, Closure $next)
{
    $user = Auth::user();
    if (!$user->is_admin()) {
        session()->flash('succsess', 'Нема прав доступу');
        return redirect()->route('index');
    }
    return $next($request);
}
```

Рис. 21 Middleware для перевірки чи є користувач адміністратором

Проміжне програмне забезпечення викликається безпосередньо перед викликом методу контролера. Якщо запит не відповідає вимогам проміжного програмного забезпечення – метод контролера і зовсім не буде викликано.

На рис. 22 відображені маршрути, що попередньо оброблюються через middleware (див. рис. 21). За логікою програмного забезпечення користувач, що не є адміністратором не може мати доступ до цих маршрутів. Middleware перевірить цю умову і в негативному випадку перенаправить користувача на головну сторінку. У позитивному випадку Middleware викликає метод next, що передає подальше управління та виконання коду до контролеру або замикаючої функції.

```php
Route::group([
    'middleware' => 'isAdmin',
    'prefix' => 'admin'
], function () {
    Route::get('/order', 'Admin\HomeController@index')->name('home');
    Route::get('/orders/{order}', 'Admin\HomeController@show')->name('order-show');
    Route::resource('categories', 'Admin\CategoryController');
    Route::resource('goods', 'Admin\ProdController');
    Route::get('props/sort', 'Admin\PropController@sort')->name('props.sort');
    Route::resource('props', 'Admin\PropController');
    Route::resource('texthtmls', 'Admin\TexthtmlController');
});
```

Рис. 22 Група маршрутів, що оброблюються через middleware

3.2.1 Моделі. Система об'єктно-реляційного відображення (ORM) Eloquent - красива і проста реалізація шаблону ActiveRecord в Laravel для роботи з базами даних. Кожна таблиця має відповідну клас-модель, яка використовується для роботи з цією таблицею. Моделі дозволяють брати дані з таблиць, а також вставляти в них нові записи. Моделі зазвичай розташовуються в папці app. Всі моделі Eloquent успадковують клас Illuminate \ Database \ Eloquent \ Model.

Найпростіший спосіб створити екземпляр моделі - за допомогою Artisan-команди make: model: php artisan make:model <Назва моделі>.

Laravel також передбачує створення міграцію та суміжною з нею моделі за допомогою Artisan-команди: artisan make:model <Назва моделі> --migration або php artisan make:model <Назва моделі> -m.

Якщо не вказати явно, яку таблицю Eloquent повинен прив'язати до моделі, то відповідно до специфікації фреймворка буде використано ім'я класу в нижньому регістрі і в множині.

У випадку на рис. 23 Eloquent припустить, що модель Prods зберігає свої дані в таблиці prods. Можна вказати довільну таблицю, визначивши властивість table в класі моделі: protected $table = 'my_ table’; .

```php
class Prods extends Model
{
    protected $fillable = [
        'name',
        'code',
        'category_id',
        'price',
        'articul',
        'description',
        'rent1',
        'rent2',
        'rent3',
        'rent4',
        'iamge',
    ];
}
```

Рис. 23 Модель товару

На рис. 24 приведений список всіх моделей, що використовувались у проекті. Розглянемо модель Category.

Рис. 24 Список моделей проекту

Модель категорій на рис. 25 містить два методи та одну захищену змінну. Захищена змінна fillable містить назви полів однойменної таблиці в базі даних, що мають бути відкриті для редагування через запити з веб-інтерфейсу.

```php
class Category extends Model
{
    protected $fillable = [
        'name',
        'code',
        'description',
        'iamge',
        'meta_t',
        'meta_k',
        'meta_d',
    ];
    public function prods()
    {
        return $this->hasMany(Prods::class);
    }
    public function props()
    {
        return $this->belongsToMany(Prop::class);
    }
}
```

Рис. 25 Модель категорій

Метод prods() вказує на зв'язок моделі, а відповідно і таблиці в базі даних, категорії з таблицею товару. Метод props() аналогічно до методу prods() вказує на зв'язок з таблицею властивостей.

Модель товарів (Prods). рис. 26 показує методи, що належать до моделі товарів (Prods). Текі методи як category(), images(), propsProds() – описують зв’язки між з моделями категорій, додаткових фото та властивостей відповідно.

```php
public function category()
{
    return $this->belongsTo(Category::class);
}

public function images()
{
    return $this->hasMany(Goods_Image::class);
}

public function imagesPath()
{
    $imgPath = $this->hasMany(Goods_Image::class);
    $array = [];
    foreach ($imgPath as $path) {
        array_push($array,  $path->iamge);
    }
    return $array;
}

public function propsProds()
{
    return $this->belongsToMany(Prop::class)->withPivot('unit')->withTimestamps();;
}

public function getPriceForCount()
{
    if (!is_null($this->pivot)) {
        return $this->rent1 * $this->pivot->count;
    } else {
        return $this->rent1;
    }
}
```

Рис. 26 Методи моделі товарів

На відміну від всіх методів даного класу, що описують зв’язки можна окремо розглянути метод propsProds(). Цей метод описує зв'язок з сполучною таблицею. За замовчуванням зведена таблиця містить лише ідентифікатори таблиць, що вона поєднує. Якщо зведена таблиця містить додаткові рядки їх потрібно вказати за допомогою метода withPivot(). Так було вказано рядок із сполучної таблиці, що містить значення властивості.

Метод getPriceForCount() (рис. 27 та рис. 28) виконує функцію калькулятора за кількістю для адміністративної панелі та кошику. При кожній зміні кількості товару користувачем цей метод перерахує вартість за актуальною кількістю. Метод викликає значення pivot (pivot – це всі додаткові рядки із зведених таблиць для конкретної моделі) та знаходить рядок count, що відповідає за кількість товару у кошику та на момент оформлення замовлення.

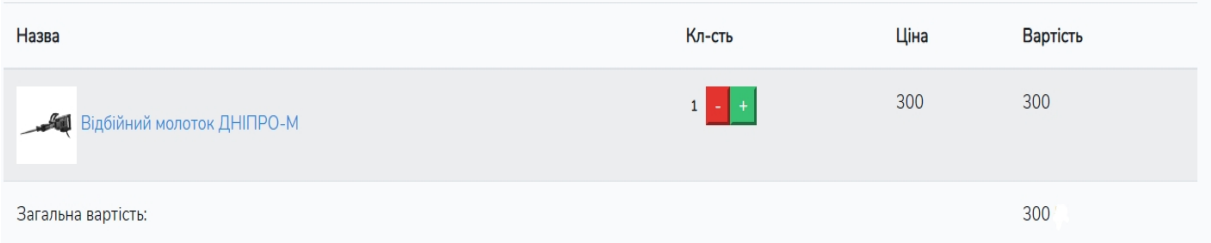


Рис. 27 Вартість товару при додаванні в накладну

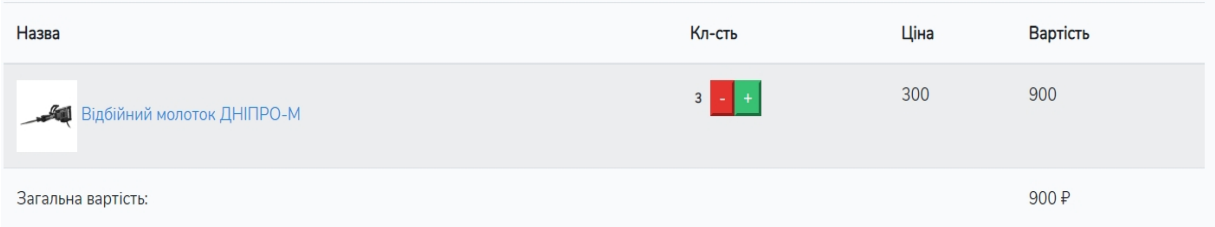


Рис. 28 Виклик getPriceForCount та зміна вартості товару

Модель додаткових фото товарів (Goods_image). До моделі на рис. 29 належить лише два елементи: масив захищених даних для редагування таблиці через запити з сайту та метод prods() – описує зв'язок з моделлю товарів.

```php
class Goods_Image extends Model
{
    protected $fillable = [
        'prods_id',
        'iamge',
    ];
    public function prods()
    {
        return $this->belongsTo(Prods::class);
    }
}
```

Рис. 29 Модель додаткових фото

Модель на рис. 30 також має масив даних, що можна редагувати. Та два методи: prodsProps() та categories() – зв’язки між моделями. Обидва методи описують зв’язки через сполучні таблиці.

```php
class Prop extends Model
{
    protected $fillable = [
        'name',
        'code',
        'unit',
    ];
    public function prodsProps()
    {
        return $this->belongsToMany(Prods::class)->withPivot('unit')->withTimestamps();
    }

    public function categories()
    {
        return $this->belongsToMany(Category::class)->withPivot('category_id');;
    }
}
```

Рис. 30 Модель властивостей

Модель користувачів (Users). Модель користувачів представляє собою модель за замовчуванням, що створюється автоматично «з коробки» під час стандартного створення Laravel проекту. Дану модель було модифіковано додатковим методом is_admin() (рис. 31). Цей метод використовується для виводу адміністративних елементів інтерфейсу (посилання в шапці сайту на сторінку панелі адміністратора, розлогінення користувача) та допуску користувача до посилань доступних лише адміністратору. Метод повертає булеве значення для рядка is_admin з таблиці користувачів для подальшої обробки.

```php
public function is_admin()
{
    return $this->is_admin == 1;
}
```

Рис. 31 Метод is_admin

Модель замовлення на рис. 32 (кошик, Order).

```php
class Order extends Model
{
    public function prods()
    {
        return $this->belongsToMany(Prods::class)->withPivot('count')->withTimestamps();
    }

    public function getFullPrice()
    {
        $sum = 0;
        foreach ($this->prods as $prod) {
            $sum += $prod->getPriceForCount();
        }
        return $sum;
    }

    public function saveOrder($name, $phone)
    {
        if ($this->status == 0) {
            $this->name = $name;
            $this->phone = $phone;
            $this->status = 1;
            $this->save();
            session()->forget('orderId');
            return true;
        } else {
            return false;
        }
    }
}
```

Рис. 32 Модель Order

Модель запиту виконує дві функції: зберігає оформлений запит та реалізує накладну. В даному проекті запит було реалізовано як не завершену накладну. При додаванні товару до запиту створюється кука в сесії – orderId. Товар, що був добавлений до запиту записується в сполучну таблицю order_prods, що в свою чергу зберігає ідентифікатор товару, значення orderId (ідентифікатор замовлення) та кількість конкретного товару. Паралельно створюється запис в таблиці orders зі статусом 0 (замовлення не оформлене). Після оформлення накладної значення статусу зміниться на 1

(накладна оформлена). Цей механізм описаний у методі saveOrder(). Метод збереження (оформлення) накладної оновлює запис додаючи до нього ім’я та номер, що вказує користувач, та оновлює запис статусу. Потім видаляє куку orderId – накладна оформлена.

Модель редагування статичного тексту (textHtml). Модель редагування тексту на рис. 33 має найпростішу структуру. Вона не містить методів вибірки або опрацювання даних оскільки елементи які може редагувати адміністратор задаються розробником. Отже модель містить лише масив даних, що може редагувати адміністратор через інтерфейс – заголовок та текст (контент).

```
class Texthtml extends Model
{
    protected $fillable = [
        'title', 'content',
    ];
}
```

Рис. 33 модель textHtml

3.2.5 Сесії (session). HTTP-програми не мають станів. Сесії - спосіб збереження інформації про користувача між окремими запитами. Laravel поставляється з безліччю різних механізмів сесій, доступних через єдиний API.

3.2.6 Запити (request) та спалахи (flash). Кожен запит (request) зроблений за допомогою веб-інтерфейсу або в веб-середовищі має метод відправлення та дані, що він відправляє. Додавання товару до кошику, зміна кількості товару у кошику, оформлення замовлення, додавання, видалення, редагування бідь- яких записів до бази даних – все це запити, що мають обов’язкові методи відправки та зазвичай відправлення запитів через інтерфейс відбувається через html-тег <form> (далі форма).

Форма має атрибут method –відповідає за метод відправки даних get | post | put | delete | head | patch.

Одним із обов'язкових та досить важливих елементів кожної форми є CSRF токен (рис. 34). Laravel вимагає наявність такого поля в кожній форми з цілю захисту від CSRF-атак. Фреймворк надає таку можливість за замовчуванням, вказавши в контейнері формі команду @csrf. Якщо команда @csrf не викликана в контейнері форми, то Laravel відхилить запит з подальшим редіректом на сторінку помилки 419 - CSRF-токен застарів або виявився некоректним.

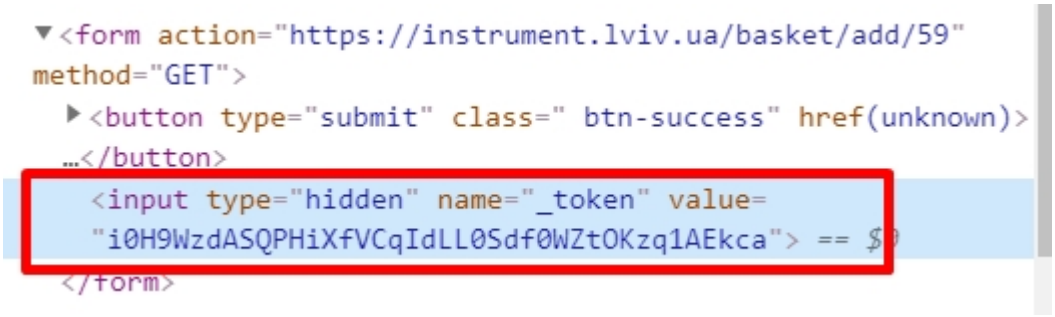


Рис. 34 Приховано поле форми створене командою @csrf

В Laravel також кожен маршрут має методи передачі.

На рис. 35 показані декілька маршрутів. URI маршруту представлені в другій колонці, методи передачі для маршруту описані в колонці номер один та назви маршрутів (допоміжний елемент, що спрощує роботу з маршрутами у коді) представлені в третій колонці.

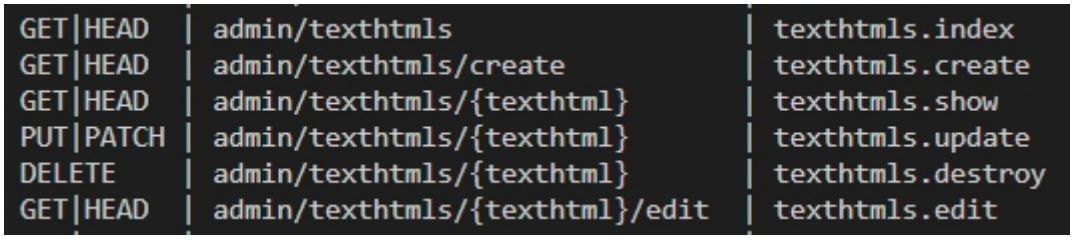


Рис. 35 Список декількох маршрутів та методи їх передачі

Це один з інструментів використання сесій – flash (дослівний переклад с англ. – спалах). Іноді потрібно вивести елементи інтерфейсу в сеансі лише для

наступного запиту. Це можна реалізувати використовуючи метод спалаху. Дані, що зберігаються в сеансі за допомогою цього методу, будуть доступні негайно та під час наступного HTTP-запиту. Після наступного HTTP-запиту такі дані будуть видалені. Дані Flash (рис. 36) в першу чергу корисні для короткочасних повідомлень про стан об’єктів.

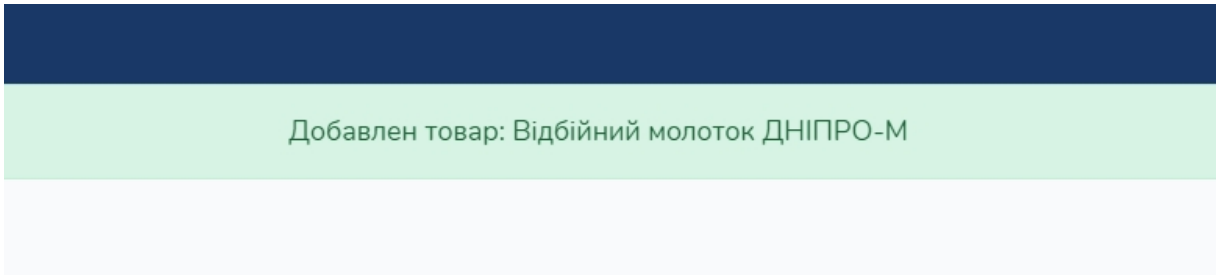


Рис. 36 Спалах-сповіщення про доданий до кошику товар

3.2.7 Взаємодії між елементами MVC. Розглянемо методи, алгоритми та механізми взаємодію у контролері, що відповідає за запит та оформлення накладної. Цей контролер містить методи для додавання товару до запиту, зміну кількості товару, ініціацію та очистку запиту, оформлення накладної.

Почнемо з методу basketAdd($goodsId) – цей метод відповідає за додавання товару до запиту. Метод має параметр – ідентифікатор товару.

В методі перевіряється ідентифікація користувача через сесію та якщо користувач не ідентифікований присвоює йому ідентифікатор.

При додаванні товару до запиту, метод перевіряє чи існує такий товар в запиті і, якщо існує – збільшить його кількість у відповідній таблиці або створить новий запис при відсутності товару. Також передбачена

ідентифікація зареєстрованого користувача. При успішному виконанні метод створює спалах (flash) та викликає перехід на сторінку кошику зі спалахом.

BasketRemove() – метод, що відповідає за видалення товару з запиту.

В методі перш за все ідентифікується користувач, якщо користувач не ідентифіковано – метод зупинить подальше виконання та перенаправить користувача на сторінку запиту яка в свою чергу за відсутності товару перенаправить користувача на головну сторінку з викликом спалаху (рис. 38 та рис. 39). Якщо користувача ідентифіковано метод буде слідкувати за кількістю товару у запиті в випадку якщо кількість менше одного і було викликано даний метод – товар буде видалено з відповідної таблиці бази даних. Після успішного виконання користувач буде направлений на головну сторінку зі сповіщенням про те, який саме товар було видалено (сповіщення через спалах).

```php
public function basketAdd($goodsId)
{
    $orderId = session('orderId');
    if (is_null($orderId)) {
        $order = Order::create();
        session(['orderId' => $order->id]);
    } else {
        $order = Order::find($orderId);
    }

    if ($order->prods->contains($goodsId)) {
        $pivotRow = $order->prods()->where('prods_id', $goodsId)->first()->pivot;
        $pivotRow->count++;
        $pivotRow->update();
    } else {
        $order->prods()->attach($goodsId);
    }
    if (Auth::check()) {
        $order->user_id = Auth::id();
        $order->save();
    }
    $goods = Prods::find($goodsId);
    session()->flash('succsess', 'Добавлен товар: ' . $goods->name);
    return redirect()->route('basket');
}
```

Рис. 38 Метод додавання товару до запиту

```php
public function basketRemove($goodsId)
{
    $orderId = session('orderId');
    if (is_null($orderId)) {
        return redirect()->route('basket');
    }
    $order = Order::find($orderId);

    if ($order->prods->contains($goodsId)) {
        $pivotRow = $order->prods()->where('prods_id', $goodsId)->first()->pivot;
        if ($pivotRow->count < 2) {
            $order->prods()->detach($goodsId);
        } else {
            $pivotRow->count--;
            $pivotRow->update();
        }
    } else {
        $order->prods()->detach($goodsId);
    }
    $goods = Prods::find($goodsId);
    session()->flash('succsess', 'Удален товар: ' . $goods->name);
    return redirect()->route('basket');
}
```

Рис. 39 Метод видалення товару з кошику

3.3 Організаційна структура програмного забезпечення

3.3.1 Діаграма пакетів. Діаграми пакетів забезпечують відмінну можливість візуального представлення залежності між частинами системи і часто використовуються для діагностики або визначення порядку компіляції.

В пакетах можуть групуватися практично будь-які елементи UML (в тому числі, і самі пакети). В пакетах можуть групуватися практично будь-які елементи UML (в тому числі, і самі пакети).

На рис. 40 зображена взаємодія пакетів інтернет-магазину.

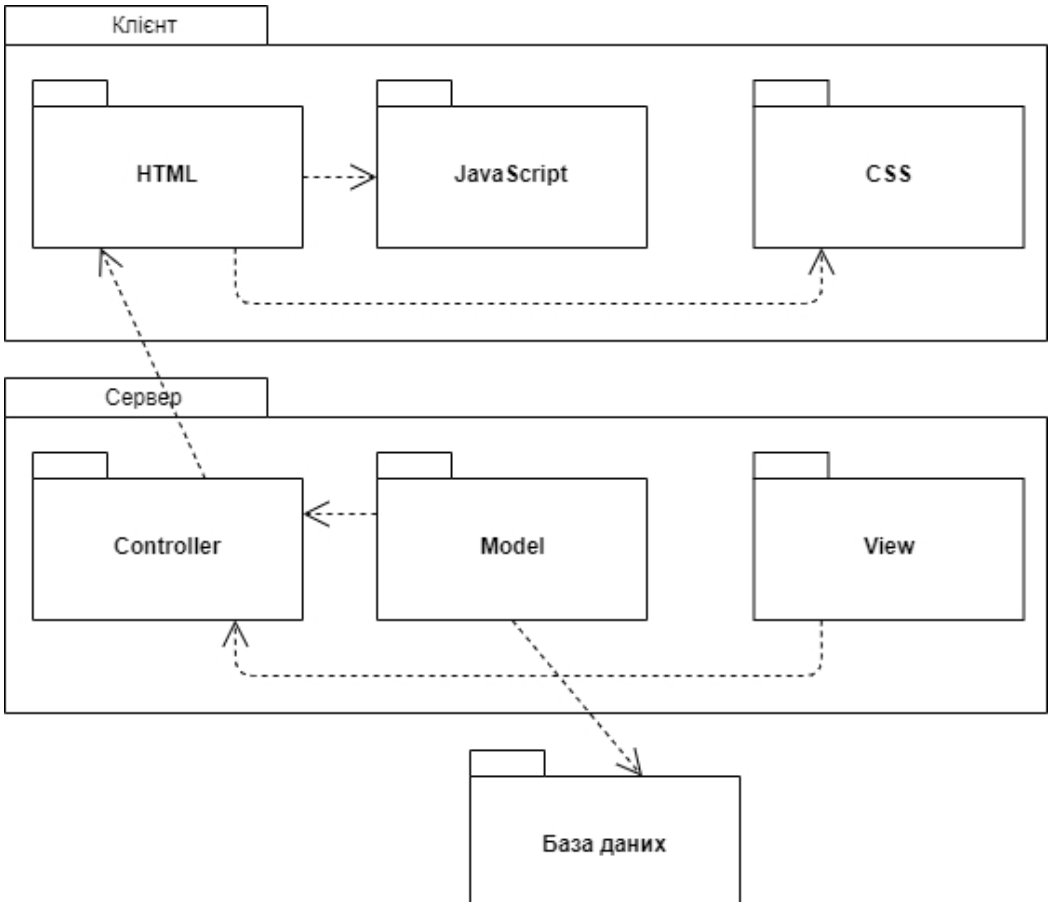


Рис. 40 Діаграма покатів

Оскільки сервер реалізує архітектуру MVC то до пакетів сервера відносим: контролер, модель, представлення та базу даних.

До клієнт браузера відносимо: HTML розмітку, JavaScript складову та CSS (мова каскадних таблиць стилів).

Controller імпортує в себе всі інші пакети серверної частини та передає дані на клієнт, а саме в HTML розмітку.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Розміщення застосунку на хостинг

Нажаль встановити застосунок без допомоги розробника буде досить проблематично, особливо для людини яка не розуміється в структурі та принципах роботи проекту.

Для початку потрібно перейти до файлового менеджеру хостингу або підключитися через FTP. Наступник кроком буде створення папки для проекту в корні хостингу.

На рис. 41 виділена папка (orest) яка була створена для копіювання проекту.

Рис. 41 Створена папка для проекту на хостингу

Наступним кроком слід створити базу даних через інструменти хостингу або інтерфейс PHPMyAdmin. Потрібно експортувати таблиці бази даних, що була створена в процесі розробки проекту, в базу даних на сервері хостнгу.

Після того як проект було скопійовано слід перейти у папку проекту та відредагувати файл .env. Параметри, що слід відредагувати:

- APP_URL – відповідає за доменне ім’я;
- DB_DATABASE – база даних на сервері;
- DB_USERNAME – ім’я користувача, що має права вносити зміни у базу даних;
- DB_PASSWORD – пароль користувача, що використовується для редагування бази даних;

Тепер шаблони сторінок та інші необхідні для відображення компоненти сайту (точка входу на розроблений проект) зберігаються за адрескою ~/orest/public/ тому перейшовши на ресурс за доменною назвою ми не зможемо побачити сайт. Щоб виправити цю ситуацію слід видалити доменну папку з хостингу (якщо домен один на хостингу, то кореневою директоріє домена матиме назву public_html або public). Наступним кроком буде створення символічного посилання (сімлінк). Посилання від директорії ~/orest/public/ до кореневої директорії домена. Для створення сімлінк виконаємо команду через термінал хостингу: ln -s ~/orest/public/ public_html. При успішному виконанні команди буде заново створена папка public_html, що буде посилатися на точку входу для розробленого проекту.

Також потрібно через термінал хостингу виконати команду: php artisan storage:link. Команда створить аліас для фото, що будуть збережені адміністратором через інтерфейс (фото товарів та категорій).

Після всіх виконаних дій можна побачити готовий проект перейшовши за посиланням. Проект готовий до роботи.

4.2 Особливості користування інтерфейсом

Структура інтерфейсу була розроблена опираючись на стандартні правила побудови UI/UX.

Кожна сторінка містить шапку сайту, що включає логотип- посилання на головну, посилання на месенджери, активні номера телефонів (при кліку на номер з мобільного пристрою номер буде відкрито для виклику), посилання на запит. Підвал сайту з посиланнями на основні сторінки першого рівня вкладеності відносно головної та копірайт.

Також більшість сторінок містять лівий сайтбар (невелике меню з важливими для користувача сторінками). Сайтбар включає посилання на категорії товарів (рис. 44).

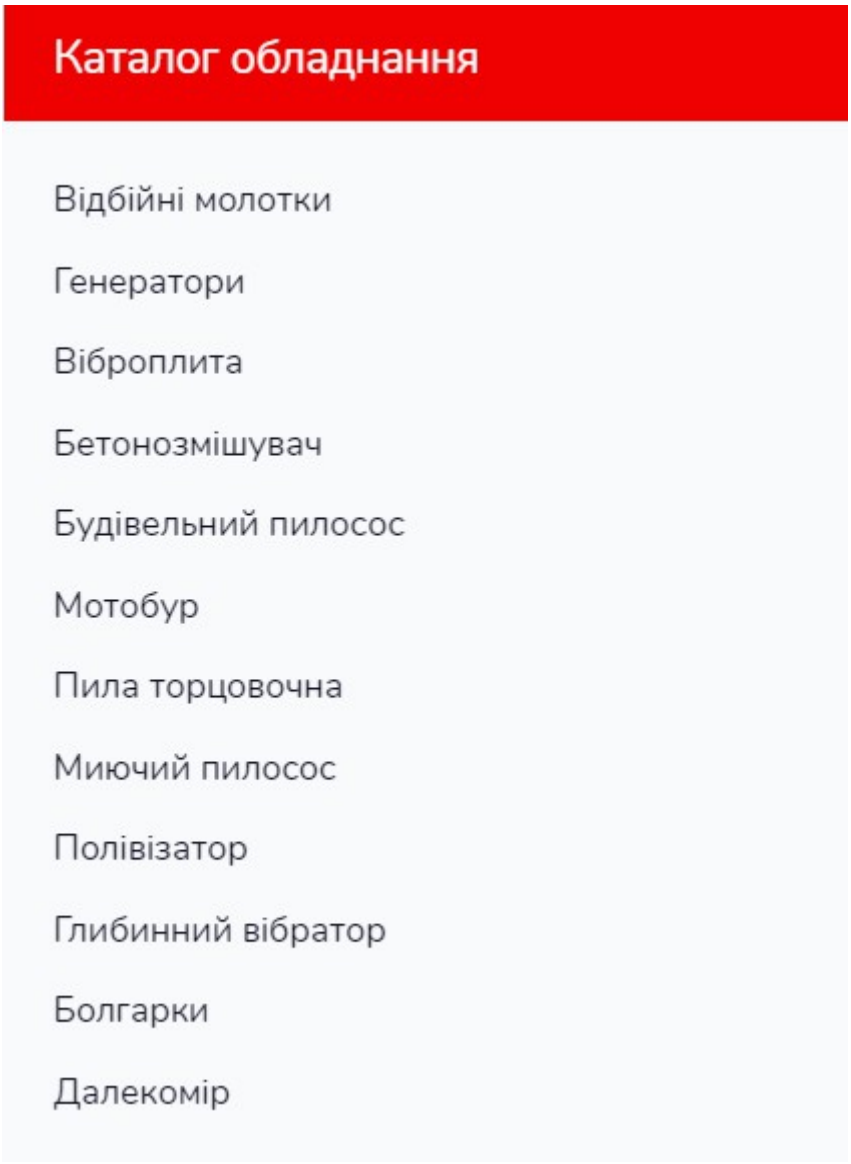


Рис. 44 Категорії товарів

На сайті розроблено три механізми оформлення накладної:

1) оформлення через телефонний зв'язок. При кліку на кнопку «набрати менеджера» відкривається спливаюче вікно з активними контактами при кліку на який номер буде відкрито для телефонного дзвінка. Такий механізм орієнтований на користувачів, що відвідують ресурс з мобільних пристроїв;
2) замовлення через кошик. Якщо покупець хоче оформити декілька товарів одночасно – розроблено механізм кошику. На картці товару або на сторінці товару слід натиснути кнопку «Орендувати». Товар буде додано до кошику. Після чого користувач зможе оформити замовлення через відповідну кнопку «оформити замовлення» (рис. 46);

3) якщо користувач не може або не хоче телефонувати та не хоче замовляти більше одного товару за раз. Зоб уникнути зайвих дій – користувачу надано можливість оформити заказ «в один клік». На сторінці товару, все що йому слід зробити – ввести номер телефону та натиснути «відправити». Менеджер отримає замовлення з поміткою «швидке замовлення» (рис. 47);

ВИСНОВКИ

В ході проведеної роботи було створено докладний опис процесу розробки програмного забезпечення інтернет-магазину з використанням архітектури MVC.

В результаті опису процесу розробки програмного забезпечення були використані діаграми описані за допомогою мови UML. Описана архітектура, бізнес-логіка, взаємодія між різними компонентами програмного забезпечення. Взаємодії між класами. Взаємодія між клієнтом та сервером.

Спроектовані діаграми дають більш ясну картину для розробки програмного забезпечення. Його функціональної складової, принципу та результату роботи.

В результаті були виконані наступні задачі:

1) проаналізована актуальність проекту на сьогоднішній час;
2) проаналізовано інтерфейс одного з найбільших інтернет-магазинів України та механізмів і принципів його робот;
3) обрані та проаналізовані методи, алгоритми, механізми та інструменти розробки інформаційної системи;
4) розроблена зручна інформаційна система, що дозволяє орендувати товар. В рамках інформаційної системи розроблені методи менеджменту ресурсом (редагування всіх важливих елементів ресурсу);
5) проаналізовано вразливості програмного забезпечення та використано сучасні методи захисту системи від цих вразливостей;
6) проаналізовано та обрано оптимальну СУБД. Створено нормалізовану базу даних;

Враховуючи все вище сказане, можна зробити висновки, що система придатна до використання та функціонування в мережі інтернет, але також може розширюватися.

Наприклад для системи додатково можуть бути розроблені модулі для внесення коментарів до товарів користувачами ресурсу. Може бути реалізований механізм оцінки товарів. Можлива розробка модулів фільтрів або сортування для товару.

Оскільки система реалізовує ООП стиль програмування - внесення додаткового функціоналу у систему не потребує кардинальних змін, або і зовсім не потребує змін, у вже функціонуючому застосунку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна документація фреймворку Laravel : [Електронний ресурс] / Режим доступу: https://laravel.com/
2. Офіційна документація MySQL : [Електронний ресурс] / Режим доступу: https://www.mysql.com/
3. MySQL. Довідник по мові [Текст] / Ред. Ю.Н. Артеменко. -М .: Видавничий дім «Вільямс», 2005. -423 с.
4. Офіційна документація PHP: [Електронний ресурс] / Режим доступу: https://www.php.net/manual/ru/book.com.php
5. Мануал по MySQL5.5- [Електронний ресурс] / Режим доступу: http://dev.mysql.com/doc/refman/5.5/en/
6. HTML довідник [Електронний ресурс] / Режим доступу: http://htmlbook.ru/
7. Етапи проектування [Електронний ресурс] - Режим доступу:

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

Сторінок – 18

Шаблон шапки та підвалу сайту

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@yield('title')</title>
    <meta name="keywords" content="@yield('keywords')">
    <meta name="description" content="@yield('descr')">
    <link rel="apple-touch-icon" sizes="180x180" href="{{Storage::url('apple-touch-
icon.png')}}">
    <link rel="icon" type="image/png" sizes="32x32" href="{{Storage::url('favicon-
32x32.png')}}">
    <link rel="icon" type="image/png" sizes="16x16" href="{{Storage::url('favicon-
16x16.png')}}">
    <link rel="manifest" href="{{Storage::url('site.webmanifest')}}">
    <link rel="stylesheet" href="/css/bootstrap.min.css">
    @if(Request::route()->getName() == "prod")
    <link rel="stylesheet" href="/css/lightslider.min.css">
    @endif
    <link rel="stylesheet" href="/css/main.css">
    <link rel="stylesheet" href="/css/app.css">
    <link rel="stylesheet" href="/css/touch-sideswipe.css">

    <style>
        * {
            box-sizing: border-box;
        }

        html,
        body {
            margin: 0;
            padding: 0;
        }
    </style>
</head>
```

```
<body>

   <header>
      <div class="container-fluid">
         <nav class="nav-container row">
            <div class="col-md-4 hidden-sm-down">
               <div class="nav-left">
                  <a href="/" class="logo">
                     <img src="{{Storage::url('ico/logo-new.png')}}" alt="">
                  </a>
               </div>
            </div>
            <div class="col-md-8 col-center hidden-sm-down">
               <a href="{{route('contacts')}}" class="phone-show">
                  <div class="svg-round">
                     <svg class="bi bi-house-door-
fill" width="1em" height="1em" viewBox="0 0 16 16"
                        fill="#193867" xmlns="http://www.w3.org/2000/svg">
                        <path
                           d="M6.5 10.995V14.5a.5.5 0 01-.5.5H2a.5.5 0 01-.5-.5v-7a.5.5 0 01.146-
.354l6-6a.5.5 0 01.708 0l6 6a.5.5 0 01.146.354v7a.5.5 0 01-.5.5h-4a.5.5 0 01-.5-.5V11c0-.25-
.25-.5-.5-.5H7c-.25 0-.5.25-.5.495z" />
                        <path fill-rule="evenodd" d="M13 2.5V6l-2-2V2.5a.5.5 0 01.5-
.5h1a.5.5 0 01.5.5z"
                           clip-rule="evenodd" />
                     </svg>
                  </div>
                  <span class="value">Київська обл.<br />Київ</span> </a>
               <div class="header-link">
                  <ul class="zakaz">
                     <li class="telegram"><a rel="nofollow" href="https://t.me/"

                           alt="Аренда Второй товар в telegram"><img src="{{Storage::url('ico/tel
egram.png')}}"
                              alt="Аренда Второй товар в вайбере"></a></li>
                     <li class="viber"><a rel="nofollow" href="viber://chat?number=%2B3809692
92020"><img
                              src="{{Storage::url('ico/viber.png')}}" alt="Аренда Второй товар в ва
йбере"></a>
```

```
                    </li>
                </ul>
            </div>
            <div class="toggle-menu-wrap">
                <div class="phone fright">
                    <a href="tel:+38 (096) 929 20 20">+38 (096) 929 20 20 </a>
                    <a href="tel:+38 (093) 929 20 20">+38 (093) 929 20 20 </a>
                </div>
                <a class="bascket" href="{{ route('basket') }}">
                    <svg class="bi bi-bucket-
fill" height="auto" width="2em" viewBox="0 0 16 16"
                        fill="currentColor" xmlns="http://www.w3.org/2000/svg">
                        <path fill-rule="evenodd"
                            d="M8 1.5A4.5 4.5 0 003.5 6h-1a5.5 5.5 0 1111 0h-1A4.5 4.5 0 008 1.5z"
                            clip-rule="evenodd" />
                        <path fill-rule="evenodd"
                            d="M1.61 5.687A.5.5 0 012 5.5h12a.5.5 0 01.488.608l-
1.826 8.217a1.5 1.5 0 01-1.464 1.175H4.802a1.5 1.5 0 01-1.464-
1.175L1.512 6.108a.5.5 0 01.098-.42z"
                            clip-rule="evenodd" />
                    </svg>
                </a>
                <div class="toggle-menu mb">
                    <span class="first"></span>
                    <span class="second"></span>
                    <span class="throuth"></span>
                </div>
            </div>

        </div>
        <div class="main-navigation">
            <div class="phone-row hidden-md-up">
                <ul class="zakaz">
                    <li class="phone">
                        <a class="phone open-modal" data-modal="rent-now" href="">
                            <svg class="bi bi-
phone" xmlns="http://www.w3.org/2000/svg" width="1em" height="1em"
                                viewBox="0 0 24 24">
                                <path
```

```
                                d="M3.59 1.322l2.844-1.322 4.041 7.89-2.725 1.341c-
.538 1.259 2.159 6.289 3.297 6.372.09-.058 2.671-1.328 2.671-1.328l4.11 7.932s-2.764 1.354-
2.854 1.396c-7.861 3.591-19.101-18.258-11.384-22.281zm1.93 1.274l-1.023.504c-
5.294 2.762 4.177 21.185 9.648 18.686l.971-.474-2.271-4.383-1.026.5c-3.163 1.547-8.262-
8.219-5.055-9.938l1.007-.497-2.251-4.398zm7.832 7.649l2.917.87c.223-.747.16-1.579-.24-
2.317-.399-.739-1.062-1.247-1.808-1.469l-.869 2.916zm1.804-
6.059c1.551.462 2.926 1.516 3.756 3.051.831 1.536.96 3.263.498 4.813l-1.795-.535c.325-
1.091.233-2.306-.352-3.387-.583-1.081-1.551-1.822-2.643-2.146l.536-1.796zm.95-
3.186c2.365.705 4.463 2.312 5.729 4.656 1.269 2.343 1.466 4.978.761 7.344l-1.84-.548c.564-
1.895.406-4.006-.608-5.882-1.016-1.877-2.696-3.165-4.591-3.729l.549-1.841z" />
                        </svg>

                    </a>
                </li>

                <li class="viber"><a rel="nofollow" href="viber://chat?number=%2B3809692
92020"><img
                            src="{{Storage::url('ico/viber.png')}}" alt="Зв'язок у Viber
"></a>
                </li>


                <li class="telegram"><a rel="nofollow" href="https://t.me/"
                        alt="Зв'язок у  telegram"><img src="{{Storage::url('ico/tel
egram.png')}}"

                            alt="Зв'язок у Viber"></a></li>

                <li class="bascket">
                    <a class="bascket" href="{{ route('basket') }}">
                        <svg class="bi bi-bucket-
fill" height="auto" width="2em" viewBox="0 0 16 16"
                            fill="currentColor" xmlns="http://www.w3.org/2000/svg">
                            <path fill-rule="evenodd"
                                d="M8 1.5A4.5 4.5 0 003.5 6h-1a5.5 5.5 0 1111 0h-
1A4.5 4.5 0 008 1.5z"
                                clip-rule="evenodd" />
                            <path fill-rule="evenodd"
                                d="M1.61 5.687A.5.5 0 012 5.5h12a.5.5 0 01.488.608l-
1.826 8.217a1.5 1.5 0 01-1.464 1.175H4.802a1.5 1.5 0 01-1.464-
1.175L1.512 6.108a.5.5 0 01.098-.42z"
```

```
                                clip-rule="evenodd" />
                        </svg>
                    </a>
                </li>
            </ul>
        </div>
        <nav class="flex phone-w100">
            <div id="touchSideSwipe" class="header-middle hidden-menu touch-side-
swipe">
                <div class="hidden-md-up">
                    <a href="/" class="logo ">
                        <img src="{{Storage::url('ico/logo-new.png')}}" alt="">
                    </a>
                    <h3 class="kategory-title"> <span>Каталог обладнання</span> </h3>
                    <ul class="kategorien-list">
                        @foreach($categories as $category)
                        <li class="last">
                            @isset($category)
                            <a class="last" href="{{ route('category', $category->code) }}"
                                title="{{$category->name}}">{{$category->name}}</a>
                            @endisset
                        </li>
                        @endforeach
                    </ul>
                </div>

                <div class="left-nav">
                    {{-- <a href="/gallery/">Продаж</a> --}}
                    {{-- <a href="{{route('ysloviya-arendi')}}">Умови</a> --}}
                    {{-- <a href="{{ route('category', 'lesa-i-vyshki-
tury') }}">Договір</a> --}}
                    <form method="POST" id="search">@csrf <input autocomplete="off" type
="search" name="q"
                            placeholder="Я шукаю">
                        <div class="q-result"></div>
                    </form>
                </div>
                <div class="right-nav">
                    <a href="{{route('dostavka')}}">Доставка</a>
```

```
                    {{-- <a href="{{route('oplata')}}">Оплата</a> --}}
                    {{-- <a href="/services/">Акции</a> --}}
                    <a href="{{route('contacts')}}">Контакти</a>
                </div>
            </div>

            {{-- @guest
            <a href="{{ route('login') }}"> Увійти </a>
            <a href="{{ route('register') }}"> Реєстрація </a>
            @endguest --}}
            @auth
            <a href="{{ route('home') }}"> Панель адміна </a>
            <a href="{{ route('get-logout') }}"> Вийти </a>
            @endauth
        </nav>
    </div>
</nav>
</div>
</header>
@if (session()->has('succsess'))
<p class="flesh alert alert-success">{{ session()->get('succsess') }}</p>
@endif

@yield('content')

<footer id="footer"> <i class="car-delivery"><a href="dostavka.html"></a></i>
<div class="inner"> <span class="copy">Made by Oleksandr ©2025</span>
<nav>
{{-- <a href="{{route('ysloviya-arendi')}}">Умови</a> --}} <a
href="{{route('dostavka')}}">Доставка</a>
{{-- <a href="{{route('oplata')}}">Оплата</a> --}}
<a href="{{route('contacts')}}">Контакти</a>
</nav>
</div>
</footer>
</body>
<div class="rent-now modal">
<div class="wrap">
```

```
        <div class="close modal-close">x</div>
        <a class="reno_phone" href="tel:+380969292020">+38 096 929 20 20</a>
        <a class="reno_phone" href="tel:+380939292020">+38 093 929 20 20</a>
    </div>
</div>
<script src="/js/touch-sideswipe.js"></script>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>

@if(Request::route()->getName() == "prod")
<script src="/js/lightslider.min.js"></script>
<script>
    $(document).ready(function() {
        $(".owl-carousel").lightSlider({
            gallery:true,
            item:1,
            auto: 1,
            loop:true,
            thumbItem: 4,
            slideMargin:0,
            enableDrag: false,
            currentPagerPosition:'left',
            onSliderLoad: function(el) {
                el.lightGallery({
                    selector: '.owl-carousel .lslide'
                });
            }
        });
    });
        $(function() {
            $('.tabs-titles .item').on('click', function(e){
                let open = $(this).data('tab');
                $(this).hasClass('active') ? null : $('.tabs-
titles .item').removeClass('active'), $(this).addClass('active'),
                                        $('.tabs-
contents .item').removeClass('active'), $(`.${open}`).addClass('active') ;
                                    })
        })
</script>
@endif
```

```
<script type="text/javascript">
    var config = {
        elementID: 'touchSideSwipe',
        elementWidth: 320, //px
        elementMaxWidth: 0.8, // *100%
        sideHookWidth: 44, //px
        moveSpeed: 0.2, //sec
        opacityBackground: 0.8,
        shiftForStart: 50, // px
        windowMaxWidth: 991.99, // px
    }
    var touchSideSwipe = new TouchSideSwipe(config);
    let form = $('#search');
    form.submit(function (e) {
        e.preventDefault();
        search()
    });
    $('input[type="search"]').keypress(function(){
        search()
    })

    $('input[type="search"]').on('change', search);

    $('input[type=search]').on('search', function () {
        search()
    });

    function search(){
        console.log($('input[type="search"]').val().length)
        if($('input[type="search"]').val().length < 3){
            $('.q-result').html('');
        }else{
            $.ajax({
            type: "POST",
            url: `{{ route('search') }}`,
            data: form.serialize(),
            headers: {
                'X-CSRF-Token': form.children("input[name=_token]").val(),
```

```
            },
            success: function(data)
            {
             let startHtml = '';
             let productsHtml = data.reduce(function(htmlAccumulator, el) {
             let divHtml = `<a href="${el.link}">${el.name} | Цена ${el.rent1} </a>`;
                 return htmlAccumulator + divHtml;
             }, startHtml);
             $('.q-result').html(productsHtml);
                console.log(data); // show response from the php script.
            }
          });
        }
    }

    $('.open-modal').on('click', function(e){
        e.preventDefault();
        let modal = $(this).data('modal');
       $(`.${modal}`).css("display", "flex");
    })

    $('.modal-close').on('click', function(e){
        e.preventDefault();
        $(this).parents( ".modal" ).css( "display", "none" );
    })

    $('.accordion .item').on('click',function(){
        $(this).toggleClass('active');
    })
</script>

</html>
```

Ресурс контролер товарів

```
<?php

namespace App\Http\Controllers\Admin;
```

```php
use App\Category;
use App\Goods_Image;
use  App\Http\Controllers\Controller;
use App\Http\Requests\ProdRequest;
use App\Prods;
use App\Prop;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Storage;
use PhpParser\Node\Stmt\Foreach_;

class ProdController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $goods = Prods::get();
        return view('auth.goods.index', compact('goods'));
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        // Prop::
        // $last_prod = Prods::latest('id')->first('id')->id + 1;
        // if (is_null($last_prod)) {
        //     $order = Prop::create();
        //     session(['orderId' => $order->id]);
        // } else {
        //     $order = Prop::find($orderId);
        // }
```

```
        $categories = Category::get();
        $category = $categories->first();
        $props = $category->props;
        // dd($props);
        return view('auth.goods.form', compact('categories', 'props'));
    }


    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(ProdRequest $request)
    {
        //$prods = Prods::create(['name' => $request->name, 'code' => $request-
>code, 'category_id' => $request->category_id, 'price' => $request->price, 'articul' => $request-
>articul, 'description' => $request->description, 'iamge' => $request->code])->id;

        $params = $request->all();
        $params['code'] = Controller::generate_chpu($params['name']);

        unset($params['iamge']);

        if ($request->has('iamge')) {
            $path = $request->file('iamge')->store('goods_images');
            $params['iamge'] = $path;
        }
        $prods = Prods::create($params)->id;

        $goods = Prods::find($prods);

        $images = [];

        if ($request->hasFile('goods_images')) {
            foreach ($request->file('goods_images') as $key => $image) {
                $images[$key] = new Goods_Image(array('iamge' => $image-
>store('goods_images')));
```

```
            $images[$key] = $goods->images()->save($images[$key]);
        }
    }

    foreach ($request->input('unit') as $key => $value) {
        $prop = Prop::findOrFail($key);
        $prop->prodsProps()->attach($prods, ['unit' => $value]);
    }

    return redirect()->route('goods.index');
}

/**
 * Display the specified resource.
 *
 * @param  \App\Prods  $prods
 * @return \Illuminate\Http\Response
 */
public function show(Prods $good)
{
    return view('auth.goods.show', compact('good'));
}

/**
 * Show the form for editing the specified resource.
 *
 * @param  \App\Prods  $prods
 * @return \Illuminate\Http\Response
 */
public function edit(Prods $good)
{

    $categories = $good->category;
    $props = $categories->props;
    $propsGood = $good->propsProds;

    return view('auth.goods.form', compact('good', 'categories', 'props', 'propsGood'));
}
```

```
    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \App\Prods  $prods
     * @return \Illuminate\Http\Response
     */
    public function update(ProdRequest $request, Prods $good)
    {
        $params = $request->all();
        $params['code'] = Controller::generate_chpu($params['name']);

        unset($params['iamge']);

        if ($request->has('iamge')) {
            Storage::delete($good->iamge);
            $path = $request->file('iamge')->store('goods_images');
            $params['iamge'] = $path;
        }

        $images = [];

        if ($request->hasFile('goods_images')) {
            foreach ($request->file('goods_images') as $key => $image) {
                $images[$key] = new Goods_Image(array('iamge' => $image-
>store('goods_images')));
                $images[$key] = $good->images()->save($images[$key]);
            }
        }

        $good->update($params);

        foreach ($request->input('unit') as $key => $value) {
            $prop = Prop::findOrFail($key);
            $prop->prodsProps()->sync($good, ['unit' => $value]);
            $prop->prodsProps()->updateExistingPivot($good, ['unit' => $value]);
        }
        return redirect()->route('goods.index');
    }
```

```
    /**
     * Remove the specified resource from storage.
     *
     * @param  \App\Prods  $prods
     * @return \Illuminate\Http\Response
     */
    public function destroy(Prods $good)
    {
        //dd($good->images()->get('iamge')->pluck('iamge')->all());

        Storage::delete($good->images()->get('iamge')->pluck('iamge')->all());
        $good->images()->delete();


        $good->propsProds()->wherePivot('prods_id', $good->id)->detach();
        Storage::delete($good->iamge);
        $good->delete();
        return redirect()->route('goods.index');
    }
}
```

Код реєстрації middleware у файлі Kernel.php

```
<?php

namespace App\Http;

use Illuminate\Foundation\Http\Kernel as HttpKernel;

class Kernel extends HttpKernel
{
    /**
     * The application's global HTTP middleware stack.
     *
     * These middleware are run during every request to your application.
     *
```

```
     * @var array
     */
    protected $middleware = [
        \App\Http\Middleware\TrustProxies::class,
        \Fruitcake\Cors\HandleCors::class,
        \App\Http\Middleware\CheckForMaintenanceMode::class,
        \Illuminate\Foundation\Http\Middleware\ValidatePostSize::class,
        \App\Http\Middleware\TrimStrings::class,
        \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,
    ];

    /**
     * The application's route middleware groups.
     *
     * @var array
     */
    protected $middlewareGroups = [
        'web' => [
            \App\Http\Middleware\EncryptCookies::class,
            \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
            \Illuminate\Session\Middleware\StartSession::class,
            // \Illuminate\Session\Middleware\AuthenticateSession::class,
            \Illuminate\View\Middleware\ShareErrorsFromSession::class,
            \App\Http\Middleware\VerifyCsrfToken::class,
            \Illuminate\Routing\Middleware\SubstituteBindings::class,
        ],

        'api' => [
            'throttle:60,1',
            \Illuminate\Routing\Middleware\SubstituteBindings::class,
        ],
    ];

    /**
     * The application's route middleware.
     *
     * These middleware may be assigned to groups or used individually.
     *
     * @var array
```

```
    */
    protected $routeMiddleware = [
        'auth' => \App\Http\Middleware\Authenticate::class,
        'isAdmin' => \App\Http\Middleware\administrator::class,
        'basketNoEmpty' => \App\Http\Middleware\BasketNoEmpty::class,
        'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
        'bindings' => \Illuminate\Routing\Middleware\SubstituteBindings::class,
        'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
        'can' => \Illuminate\Auth\Middleware\Authorize::class,
        'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
        'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
        'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
        'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
        'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
    ];
}
```

Маршрути програмного забезпечення

```
<?php

use Illuminate\Support\Facades\Route;

    /*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/;
Auth::routes([
    'reset' => false,
    'confirm' => false,
    'verify' => false
```

```
]);

Route::get('/logout', 'Auth\LoginController@logout')->name('get-logout');

Route::group([
    'middleware' => 'auth'
], function () {
    Route::post('/getprops', 'MainController@getProps')->name('getProps');
    Route::post('/delimage', 'MainController@delMoreImg')->name('delMoreImg');
    Route::group([
        'middleware' => 'isAdmin',
        'prefix' => 'admin'
    ], function () {
        Route::get('/order', 'Admin\HomeController@index')->name('home');
        Route::get('/orders/{order}', 'Admin\HomeController@show')->name('order-show');
        Route::resource('categories', 'Admin\CategoryController');
        Route::resource('goods', 'Admin\ProdController');
        Route::get('props/sort', 'Admin\PropController@sort')->name('props.sort');
        Route::resource('props', 'Admin\PropController');
        Route::resource('texthtmls', 'Admin\TexthtmlController');
    });
});


Route::get('/', 'MainController@index')->name('index');
Route::get('/ysloviya-arendi', 'MainController@showStatic')->name('ysloviya-arendi');
Route::get('/oplata', 'MainController@showStatic')->name('oplata');
Route::get('/contacts', 'MainController@showStatic')->name('contacts');
Route::get('/dostavka', 'MainController@showStatic')->name('dostavka');
Route::post('/search', 'MainController@search')->name('search');
Route::post('/delivery', 'MainController@delivery')->name('delivery');
Route::get('/categories', 'MainController@categories')->name('categories');
Route::get('/basket/add/{id}', 'BasketController@basketAdd')->name('basket-add');
Route::post('/basket/order/{id}', 'BasketController@oneClickOrder')->name('one-click-order');

Route::group(['middleware' => 'basketNoEmpty'], function () {
    Route::get('/basket', 'BasketController@basket')->name('basket');
    Route::get('/basket/order', 'BasketController@order')->name('order');
```

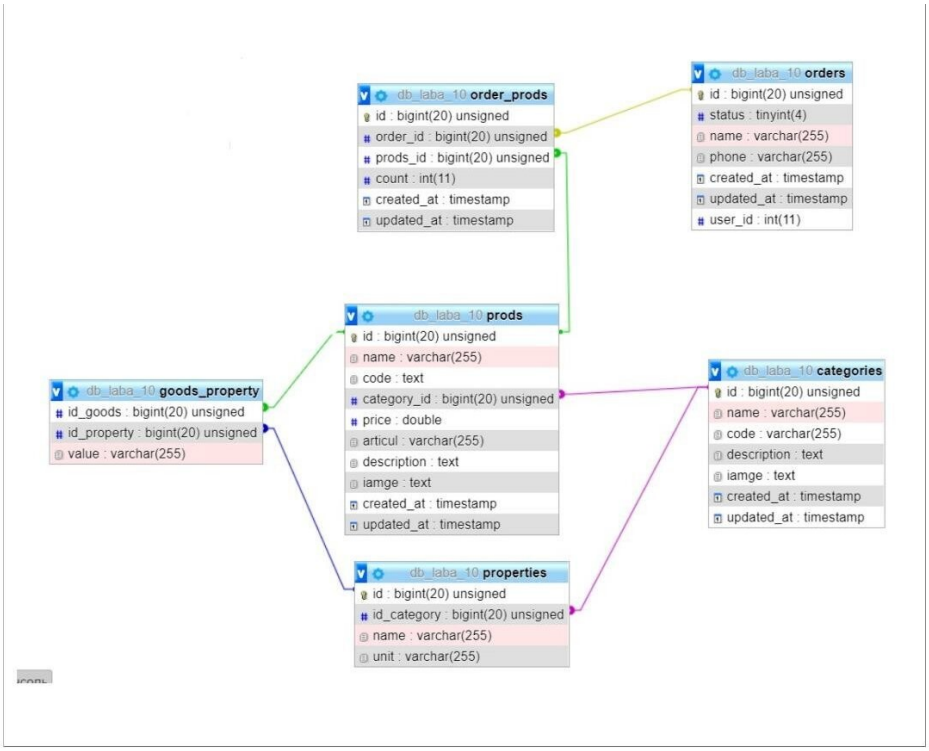
```
    Route::get('/basket/remove/{id}', 'BasketController@basketRemove')->name('basket-
remove');
    Route::post('/basket/order', 'BasketController@confirm')->name('basket-confirm');
});

Route::get('/{category}', 'MainController@category')->name('category');
Route::get('/{category}/{product}', 'MainController@prod')->name('prod');
```

ДОДАТОК Б

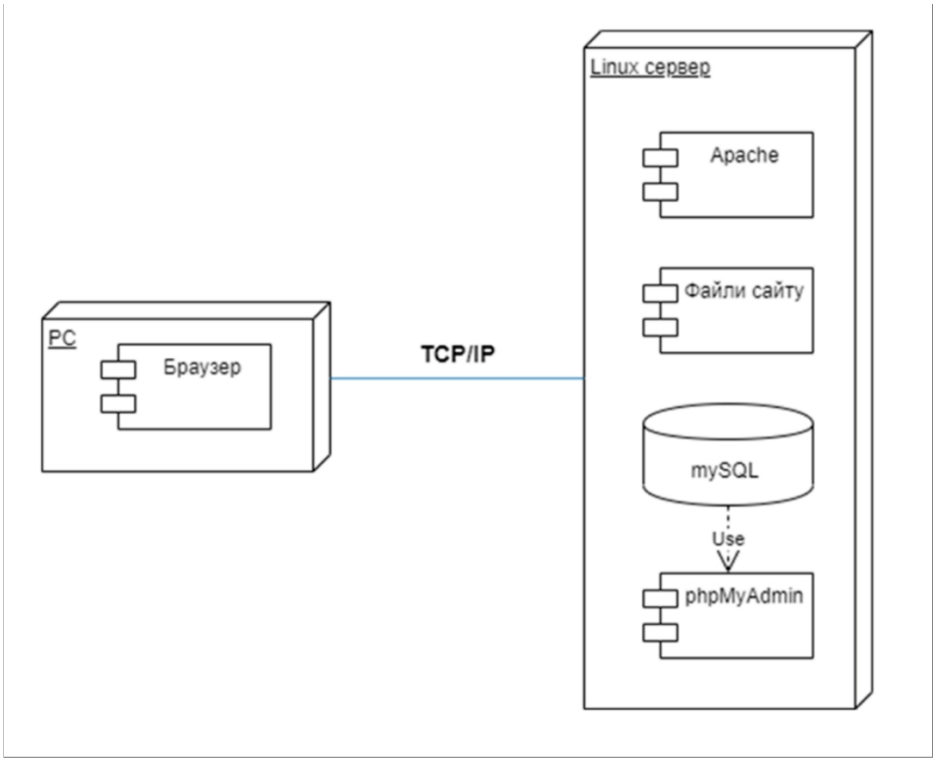
ПЛАКАТИ

Сторінок - 4

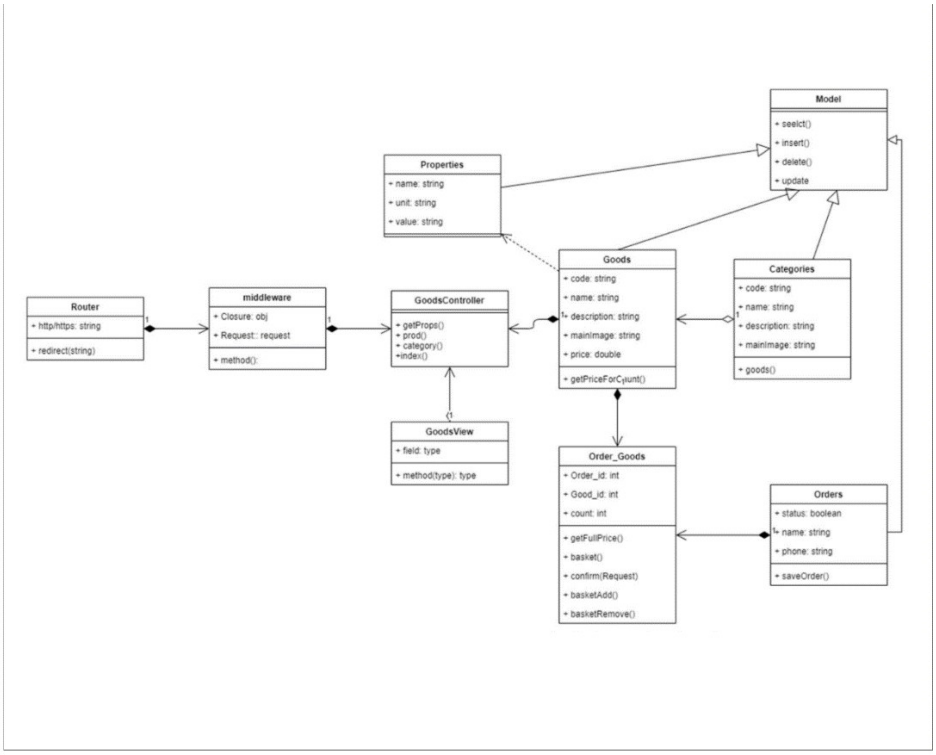
Сторінка – 1


db_laba_10 order_prods
id : bigint(20) unsigned
order_id : bigint(20) unsigned
prods_id : bigint(20) unsigned
count : int(11)
created_at : timestamp
updated_at : timestamp
db_laba_10 orders
id : bigint(20) unsigned
status : tinyint(4)
name : varchar(255)
phone : varchar(255)
created_at : timestamp
updated_at : timestamp
user_id : int(11)
db_laba_10 prods
id : bigint(20) unsigned
name : varchar(255)
code : text
category_id : bigint(20) unsigned
price : double
articul : varchar(255)
description : text
iamge : text
created_at : timestamp
updated_at : timestamp
db_laba_10 goods_property
id_goods : bigint(20) unsigned
id_property : bigint(20) unsigned
value : varchar(255)
db_laba_10 categories
id : bigint(20) unsigned
name : varchar(255)
code : varchar(255)
description : text
iamge : text
created_at : timestamp
updated_at : timestamp
db_laba_10 properties
id : bigint(20) unsigned
id_category : bigint(20) unsigned
name : varchar(255)
unit : varchar(255)

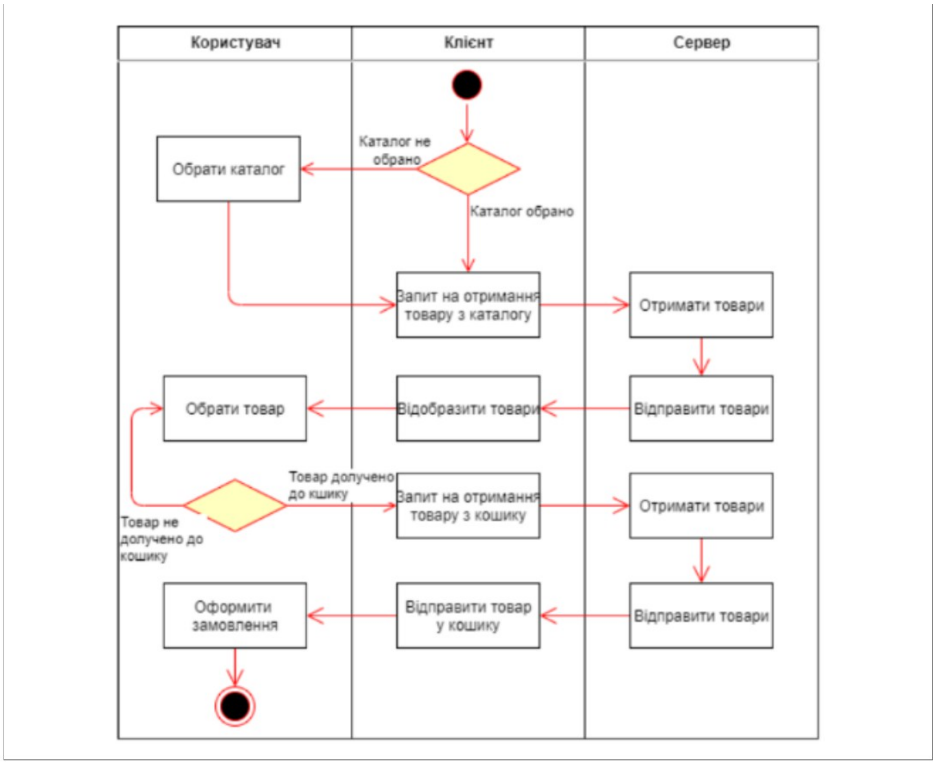
Сторінка – 2


Linux сервер
Apache
Файли сайту
PC
Браузер
TCP/IP
mySQL
Use
phpMyAdmin

Сторінка – 3


Model
+ seelct()
+ insert()
+ delete()
+ update
Properties
+ name: string
+ unit: string
+ value: string
Goods
+ code: string
+ name: string
+ description: string
+ mainImage: string
+ price: double
+ getPriceForCyiunt()
Categories
+ code: string
+ name: string
+ description: string
+ mainImage: string
+ goods()
Router
+ http/https: string
+ redirect(string)
middleware
+ Closure: obj
+ Request:: request
+ method():
GoodsController
+ getProps()
+ prod()
+ category()
+index()
GoodsView
+ field: type
+ method(type): type
Order_Goods
+ Order_id: int
+ Good_id: int
+ count: int
+ getFullPrice()
+ basket()
+ confirm(Request)
+ basketAdd()
+ basketRemove()
Orders
+ status: boolean
+ name: string
+ phone: string
+ saveOrder()

Сторінка – 4


Користувач
Клієнт
Сервер
Обрати каталог
Каталог не обрано
Каталог обрано
Запит на отримання товару з каталогу
Отримати товари
Обрати товар
Відобразити товари
Відправити товари
Товар долучено до кшику
Товар не долучено до кошику
Запит на отримання товару з кошику
Отримати товари
Оформити замовлення
Відправити товар у кошику
Відправити товари