МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

Кваліфікаційна робота бакалавра

КОМПОНЕНТНИЙ ПІДХІД ПРИ ОРГАНІЗАЦІЇ РОЗРОБКИ БАГАТОСТОРІНКОВОГО ДОДАТКУ НА ПРИКЛАДІ СТВОРЕННЯ ВЕБ-САЙТУ ДЛЯ ПІДТРИМКИ ДІЯЛЬНОСТІ МЕРЕЖІ «FANZOO»

Здобувач освіти гр. ІН-81 Вадим СИВОКОНЬ

Науковий керівник,
кандидат ф.-м. наук, доцент Олена ПРОЦЕНКО

Завідувач кафедри,
доктор технічних наук, професор Анатолій ДОВБИШ

Суми 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую _____
Зав. кафедри Довбиш А.С.
"_____" _____2022 р.

ЗАВДАННЯ

до кваліфікаційної роботи бакалавра

Здобувача освіти 4-го курсу, групи ІН-81 спеціальності 122 — Комп'ютерні науки, денної форми навчання Сивоконя В.В.

Тема: Компонентний підхід при організації розробки багатосторінкового додатку на прикладі створення веб-сайту для підтримки діяльності мережі «FanZoo»

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) інформаційний огляд; 2) вибір методу рішення; 3) програмна реалізація

Дата видачі завдання "_____"_____ 2022 р.
Керівник випускної роботи _____Проценко О.Б.
Завдання прийняв до виконання _____Сивоконь В.В.

РЕФЕРАТ

Записка: 127 стор., 10 рис., 21 табл., 7 додатків, 16 джерел.

Об’єкт дослідження — компонентний підхід при організації розробки багатосторінкового додатку.

Мета роботи — розробка програмної реалізації веб-додатку для підтримки діяльності мережі зоомагазинів «FanZoo» із застосуванням компонентного підходу.

Методи дослідження — теоретичний аналіз та узагальнення даних, отриманих зі спеціалізованих джерел, верстка веб-сторінок із застосуванням технологій HTML і CSS, програмування на мовах PHP і JavaScript, використання інструментів Composer, Laravel Mix, SASS, Node.js, npm, Faker, Fantasticon, Laravel Debugbar, робота з базою даних MariaDB.

Результати — створений веб-додаток, що здатен підтримувати діяльність мережі зоомагазинів «FanZoo». Під час розробки була сформована концепція компонентного підходу, що була втілена у вигляді невеликого JavaScript-фреймворку, за допомогою якого були реалізовані основні складові веб-сайту.

ВЕБ-ДОДАТОК, JAVASCRIPT, PHP, LARAVEL, ФРЕЙМВОРК, КОМПОНЕНТ, ПІДХІД

ЗМІСТ

ВСТУП

Ефективне керування бізнесом у наш час — це не лише планування, аналіз ринку та контроль внутрішніх процесів. Не менш важливу роль відіграє гнучкість, що виявляється у спроможності підлаштуватися під сучасні тенденції.

На сьогоднішній день вони такі, що все поступово переходить у віртуальне середовище: починаючи від спілкування в месенджерах та проведення зустрічей в онлайн-форматі, закінчуючи появою метавсесвітів (на прикладі розробок компаній Meta, Mozilla у цьому напрямі) та інтеграцією документообігу у вигляді «держави у смартфоні» застосунку «Дія». Ми є свідками цих трансформацій — вони відбуваються перед нами та змінюють світ на краще — у бік технологічного майбутнього. Прийняття цих змін та рух у ногу з часом тягне за собою виключно переваги.

З огляду на те, що люди почали приділяти увагу можливості придбання товару через Інтернет — у сучасних реаліях створення сайтів електронної комерції стало розповсюдженим явищем. Власники розглядають появу власного магазину в глобальній мережі як спосіб просування бренду, що матиме в майбутньому ймовірне збільшення прибутків.

Ведення онлайн-торгівлі надає нові можливості як для покупця, так і для власника магазину. Клієнт може ознайомитися з описом товару та його основними властивостями, а також отримати консультацію з питань, що становлять для нього зацікавленість, використовуючи контактні дані, які зазвичай зазначаються на інформаційних сторінках магазину. Також з’являється можливість уникнути витрат часу, оскільки Інтернет-магазин працює цілодобово та без вихідних, що дозволяє створити замовлення у будь-який момент часу не виходячи з дому. Завдячуючи спроможностям сучасних веб-технологій та вдалим рішенням, що приймаються під час етапу проектування, користувач отримує ефективну навігацію сайтом та пошук, що робить пошук необхідного йому товару простою задачею, а оформлення замовлення — швидким.

У свою чергу власник стає більш конкурентоспроможним, бо з’явиться опція запропонувати продукцію за нижчими цінами, аніж у звичайному магазині, що пов’язано зі зменшенням витрат на орендування приміщень та скороченням штату працівників. Поява магазину в мережі може стати основою для проведення рекламної кампанії, що дозволить збільшити клієнтську базу. Власник зможе контролювати бізнес шляхом використання ефективних інструментів адміністрування та збору аналітики.

Актуальність роботи полягає в затребуваності використання Інтернет-магазину як сучасного інструменту ведення бізнесу.

Метою роботи є проектування та розробка програмної реалізації веб-додатку для підтримки діяльності мережі зоомагазинів «FanZoo».

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Актуальність створення веб-додатків для магазинів

Веб-магазин — це Інтернет-магазин, створений за схемою B2C (бізнес для споживача) та призначений для ведення роздрібної торгівлі та продажу послуг [1].

На сьогоднішній день їх використання в якості невід'ємної складової ведення бізнесу є поширеною практикою з огляду на отримувані переваги.

Перш за все — це економічна вигода: зі збільшенням клієнтської бази матимемо зростання прибутковості ведення бізнесу. Додатково власники отримують опцію зниження витрат завдяки тому, що можуть відмовитися від орендування приміщень та скоротити штат працівників.

Інтернет-магазини мають широкі можливості з автоматизації. На веб додаток покладається виконання рутинних дій, що спрощує ведення бізнесу. Наприклад, не треба проводити інвентаризацію складських приміщень, щоб підрахувати залишки продукції після продажів. Цим може зайнятися вбудований алгоритм, що буде обчислювати нову кількість одиниць товарів одразу після оновлення статусу відповідного замовлення.

Виконані зміни в параметрах товарів будуть одразу відображені для користувачів. Таким чином не будуть відбуватися прикрі ситуації, подібно тим, як це буває в реальних магазинах, коли персонал не встигає оновити цінники.

Для зручності використання веб-додатку може бути передбачений модуль для здійснення масового завантаження даних із файлів, що надаються постачальниками, з поширеними розширеннями виду csv або xls. Це звільнить керівництво магазину від виконання об'ємної однотипної роботи.

Вдалі рішення під час проектування інтерфейсу веб-сайту дозволять надати ефективний інструмент керування бізнесом, у якому наявні всі основні функції.

Не менш важливою перевагою є зручність для користувачів, що може бути виражена в безлічі аспектів використання Інтернет-магазину.

Однією з найбільш важливих складових є вдалість реалізації елементів навігації та забезпечуваний рівень ефективності алгоритму пошуку сайтом. Це безпосередньо впливає на те, наскільки швидко користувачу вдасться знайти необхідний йому товар.

Інтернет-магазин працює цілодобово та без вихідних, що дозволяє клієнту оформити замовлення в будь-який зручний для нього час.

Якщо адміністрація сайту приділяє належну увагу актуалізації інформації, то користувач матиме з цього додаткову зручність. Як приклад: в разі надходження продукції, що раніше була відсутньою, на склад — це повинно бути зафіксовано, щоб користувач мав поточну інформацію про наявність товару в продажу.

Зазвичай розробники веб-магазинів намагаються надати користувачу якомога більше інформації про товар та супроводжуючих підказок, що допоможуть людині зробити вибір. Також можуть пропонуватися схожі або супутні продукти, що обираються вбудованими в систему спеціалізованими алгоритмами, задачею функціонування яких є допомога користувачу уникнути виконання додаткових пошуків. Іноді на такі товари навіть пропонується знижка, що робить пропозицію ще більш привабливою.

Черговою перевагою для власника є те, що поява бренду в глобальній мережі Інтернет є потужним поштовхом для його подальшого просування, оскільки слугуватиме своєрідною рекламою. У разі застосування механізмів пошукової оптимізації досягти бажаних результатів вдасться навіть швидше.

У разі використання в бізнесі веб-магазину, власник матиме можливість збирати цінну інформацію, що стане корисною для подальшого розвитку продажів. Першим способом це зробити — використовувати форму зворотного зв’язку, що має на меті отримання повідомлень від користувачів з подяками або ж зауваженнями та пропозиціями. Зазвичай такі форми заповнюються більш охоче, аніж «книга скарг та пропозицій», що відійшла в минуле. Також є можливість використання аналітичних сервісів, які збирають найрізноманітнішу ін-

формацію про клієнтів. Наприклад — звідки вони потрапили на сайт, що дозволяє зробити висновки щодо ефективності витрачання коштів на проведення тієї чи іншої рекламної кампанії. Інший варіант — оцінка конверсії, що допоможе виявити можливі проблеми та на якому кроці вони трапляються статистично частіше за все.

1.2 Аналіз існуючих Інтернет-магазинів зоотоварів

Із метою подальшого формування постановки задачі були проаналізовані декілька Інтернет-магазинів зі схожою тематикою. Це дозволить у першу чергу отримати уявлення про основні складові веб-додатків та підтримувані функції, виокремити серед них найбільш затребувані для реалізації. Потім ці елементи будуть ретельно проаналізовані на предмет вдалості реалізації, завдяки чому зможемо перейняти гарний досвід із додаванням власних покращень, а також уникнути побачених недоліків.

Почнемо з веб-магазину «MasterZoo», що представлений на рис. 1.1. На головній сторінці сайту можемо бачити насичене елементами керування меню. У верхній його частині наявні посилання на інформаційні сторінки, відсортовані за зниженням ступеня важливості: на початку — ті, що повідомляють про працюючі зоомагазини мережі, особливості надання обслуговування. Після них — відгуки, інформація про мережу, блог та інші.

Нижче представлений великий логотип, поряд із яким — пошуковий рядок, натискання на який призводить до появи модального вікна для введення запиту. Завдяки використанню асинхронних запитів до сервера результати оновлюються по мірі вводу. Внутрішній алгоритм здійснює контекстний пошук, завдяки чому користувач отримує інформацію не лише щодо товарів, а й категорій, у яких вони знаходяться. Надається можливість замовити дзвінок.

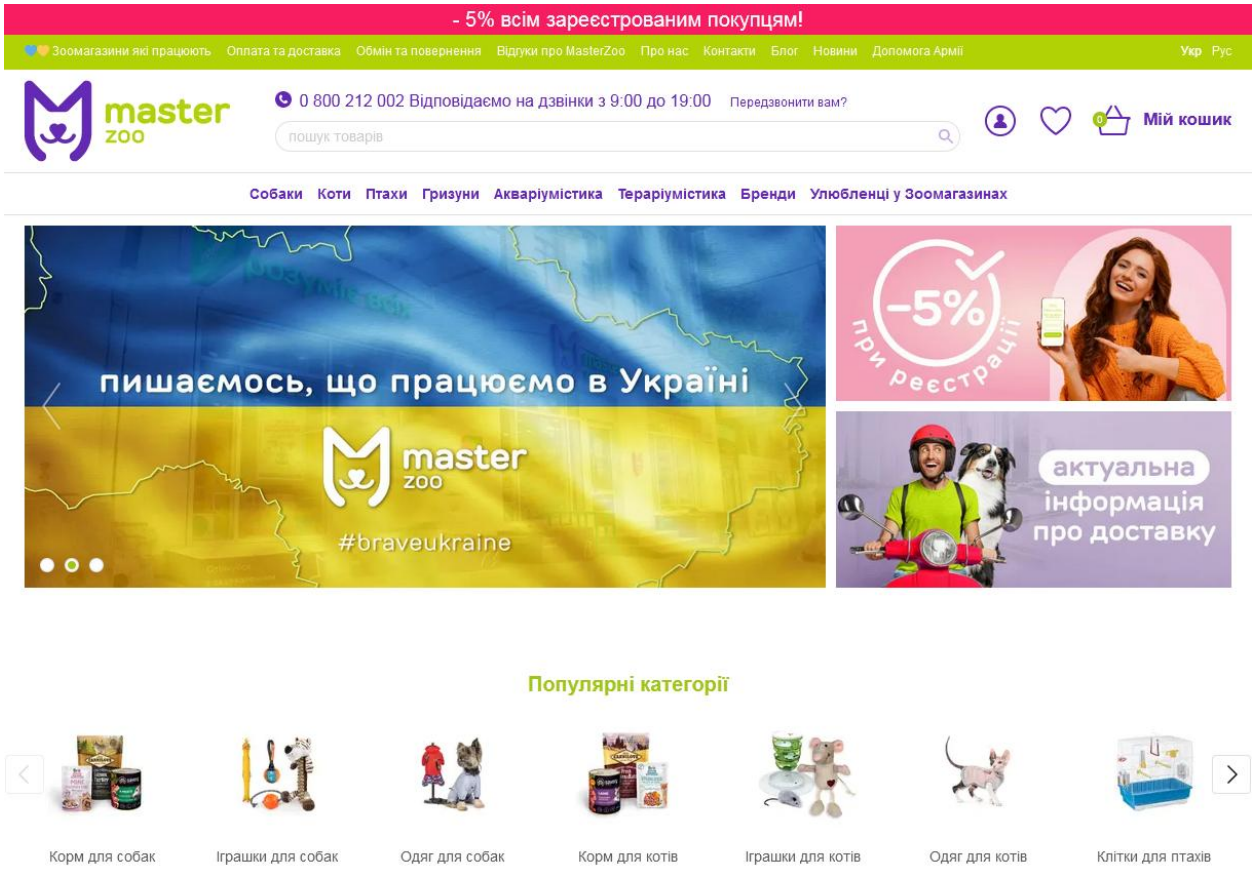

Рисунок 1.1 — Головна сторінка магазину «MasterZoo»

Окремо вирізняється набір елементів керування, що відповідають за користувацький інтерфейс: профіль, бажання та кошик.

Нижче розташований слайдер, задача якого транслювати останні новини з роботи мережі. Справа — допоміжні зображення. Нижче представлені популярні категорії, для позначення яких використовуються фотографії відповідних товарів, що їх представляють.

Внизу розташований опис мережі та підвал сайту із посиланнями на категорії товарів, їх уміст, інформаційні сторінки з контактними даними. Є можливість перемикання між мобільною та повною версіями Інтернет-магазину.

Сайт володіє зручною навігацією каталогом: передбачені можливості фільтрації списку товарів за їх властивостями, зокрема — постачальником, відображаються так звані «хлібні крихти» — шаблон навігації, який показує кожен рівень ієрархії, що веде до сторінки, на якій знаходиться відвідувач [2, с. 5]. Фіксується перелік переглянутих товарів, а також пропонуються схожі або

пов’язані, утім про погодження із використання cookie не згадується. На довгих сторінках автоматично з’являється кнопка для переміщення вгору.

Також були виявлені недоліки цієї реалізації веб-магазину. Першим є те, що тип пристрою клієнта визначається сервером. Від цього залежить, яку версію сайту буде повернуто — мобільну або повну. З огляду на цю особливість функціонування адаптивність не була реалізована повністю, про що може свідчити відсутність стиснення, приховання або ж перебудови положення елементів в разі зменшення вікна браузера. Загальна горизонтальна смуга прокрутки починається з’являтися, коли ширина viewport’у стає меншою за 1000 пікселів.

Також веб-магазин «MasterZoo» застосовує в мобільній версії сайту незвичні для користувацького досвіду елементи. Ними є використання чисельної кількості горизонтальних смуг прокрутки. Більш очікуваним дизайнерським рішенням було б зменшення розміру елементів або їх перебудова всередині контейнера.

Кнопка додавання товару в список бажань наявна, але нею неможливо скористатися без проходження авторизації на сайті. Було б краще її приховати зовсім. Інший варіант — надати цю можливість і у якості сховища використовувати cookie або сесію.

Магазин підтримує встановлення значень параметрів товарів, але не має опції здійснення порівняння.

З огляду на використання платформи Horoshop веб-додаток має гарний дизайн, але не унікальний.

Наступний веб-сайт для проведення аналізу є Інтернет-магазин «ZooBonus» (див. рис. 1.2). Дизайн, що використовується, здається застарілим, утім у ньому представлені всі основні елементи, що функціонують.

Зверху можемо бачити елемент навігації, що веде на сторінки авторизації та реєстрації. Незвичною особливістю є довгий перелік ролей, одну з яких можна обрати під час створення нового акаунта. Серед них: власник домашнього улюбленця, розподільник, клуб, Інтернет-зоомагазин, зоомагазин, ветеринарна

аптека та інші. У залежності від обраного типу користувача виводиться форма із відповідним набором полів для заповнення. Маємо таку варіативність, бо «ZooBonus» займається не лише роздрібною торгівлею, а й оптовими постачаннями для організацій різного типу.

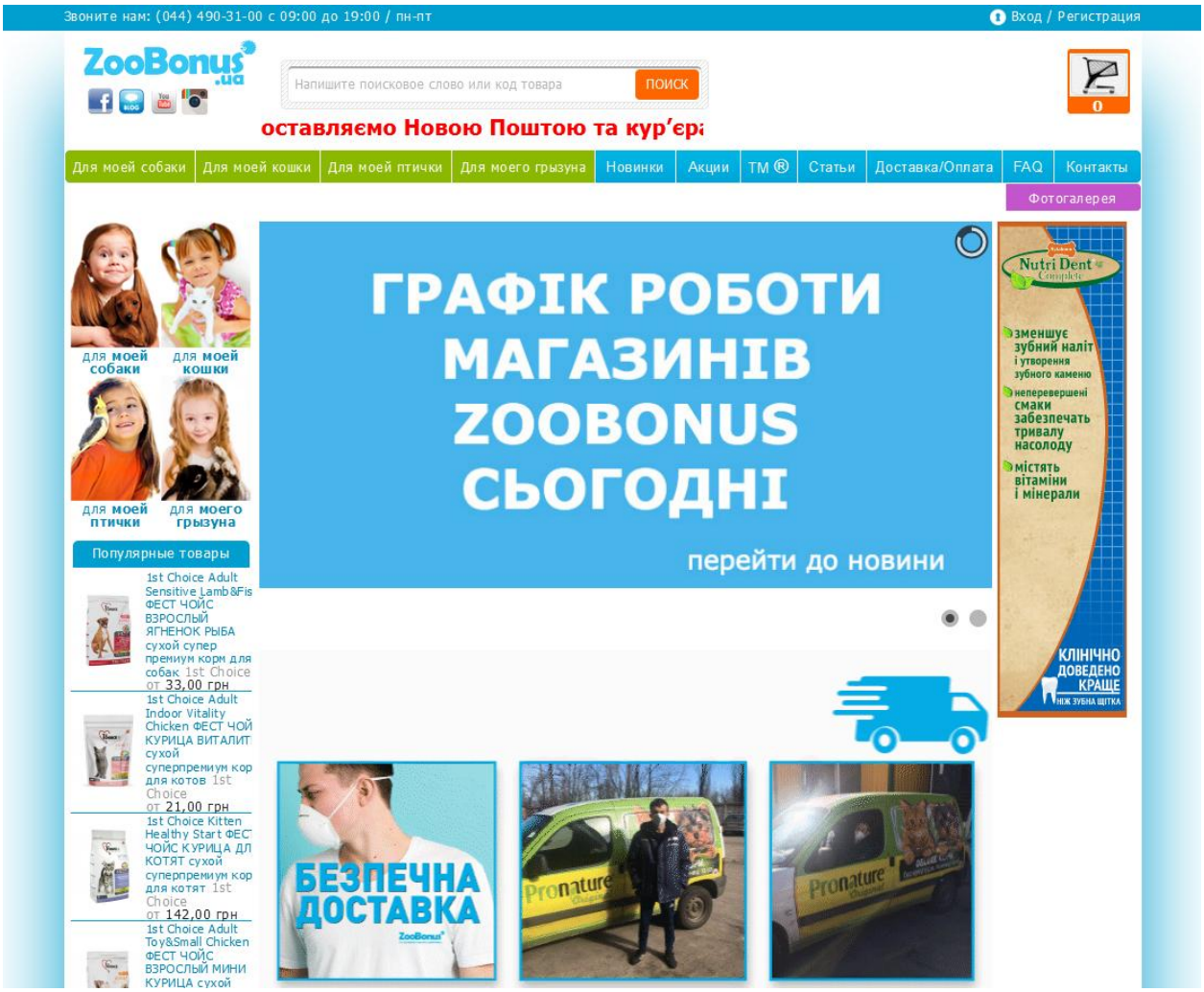

Рисунок 1.2 — Головна сторінка магазину «ZooBonus»

Пошуковий рядок, що розміщений поряд із логотипом, може надавати користувачу результати запиту на окремій сторінці. Він не використовує всього наявного вільного простору області заголовку сторінки. Справа — корзина, на якій позначається кількість товарів у кошику на поточний момент часу.

Нижче знаходяться елементи меню для надання можливості ознайомитися з каталогом. Вони позначені зеленим кольором. З огляду на велику кількість розділів та використання малого розміру шрифту важко одразу зорієнтуватися та обрати шукану категорію. У свою чергу справа представлені елементи керу-

вання блакитного кольору, що відповідають за навігацію між інформаційними сторінками. На сайті наявні «хлібні крихти», що поліпшують виконання переходів.

Дизайнери намагалися використати простір з боків, про що свідчить наявність елементів навігації та реклами в цих областях. У центрі ж розміщений слайдер — для інформування щодо останніх новин.

Однією з особливостей сайту, що розглядається, є надання користувачам можливості заповнювати галерею фотографіями з їх домашніми улюбленцями. Нижче, під нею — блок останніх відгуків користувачів та перелік постачальників продукції. За ними розміщений текст — детальний опис зоомагазину. Унизу сторінки — підвал сайту, у якому дублюються навігаційні елементи на категорії товарів та найбільш важливі сторінки. Також пропонується можливість переглянути мобільну версію сайту.

У ході аналізу складових елементів веб-магазину «ZooBonus» виявилося, що найбільш суттєвими проблемами цієї реалізації є дизайн та відсутність адаптивності. Контейнер у повній версії сайту не зменшується, тому, коли ширина viewport'у менша за 1000 пікселів, починає з'являтися горизонтальна смужка прокручування.

Сайт «zootovary.com» — ще один існуючий аналог (див. рис. 1.3). На початку перегляду одразу примітним є те, що значна частина звичної заголовної частини сайту була виокремлена в бокове меню. Напевно, дизайнери керувалися метою поліпшити навігацію, оскільки елементи для здійснення переходів постійно супроводжуватимуть користувача, незалежно від положення перегляду сторінки.

У заголовній частині залишився пошуковий рядок, контактний телефон, опції з порівняння та формування обраного. Також можна відкрити форму авторизації або перейти на сторінку реєстрації. У кутку є корзина, що в разі наповнення показуватиме користувачу кількість одиниць товарів усередині.

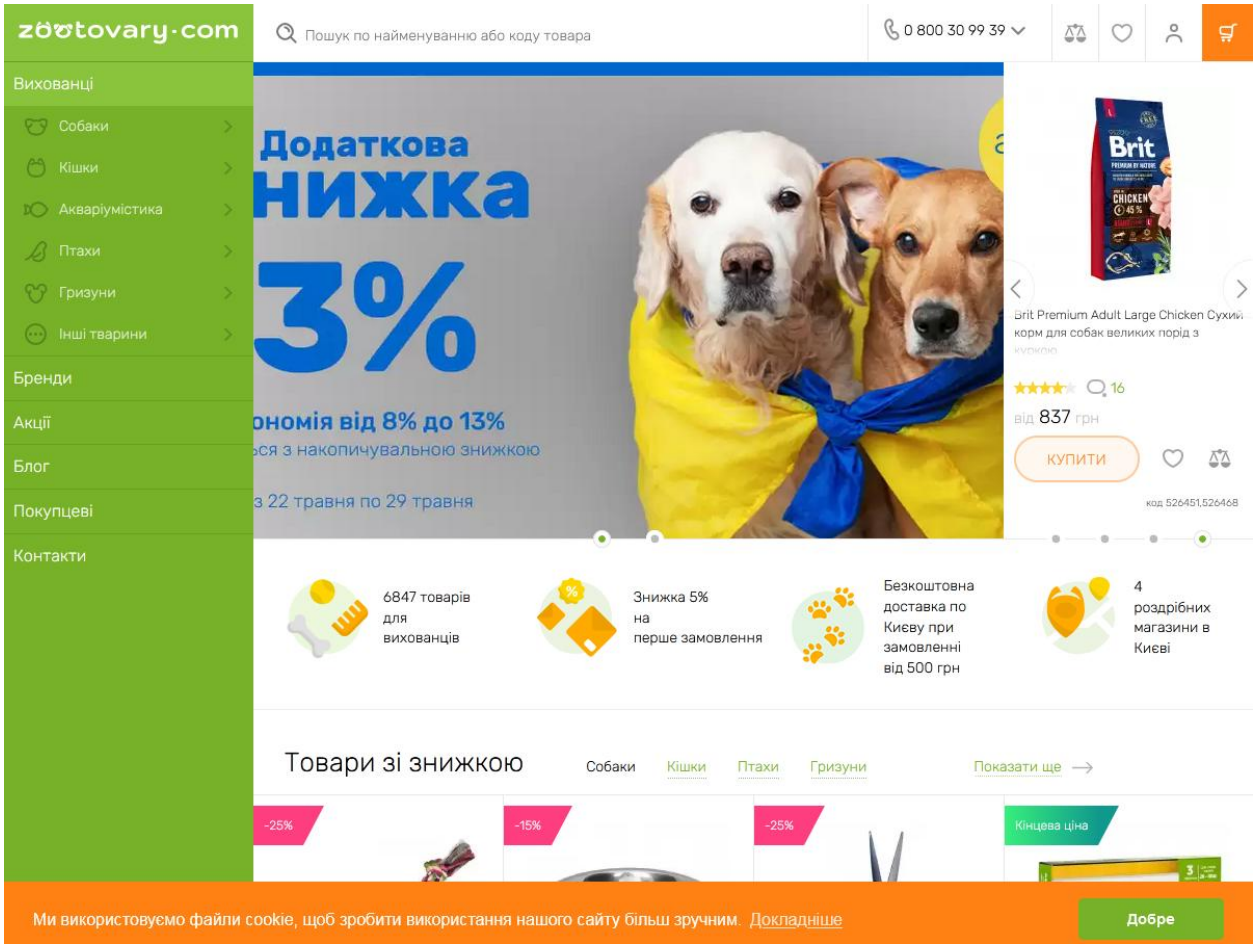

Рисунок 1.3 — Головна сторінка сайту «zootovary.com»

Дизайнери використали можливості інфографіки з метою покращення сприйняття контенту користувачами. Це помітно за іконками, що використовуються для позначення категорій товарів та кількісних переваг магазину.

У разі здійснення пошуку частина результатів буде відображена після завершення вводу рядка запиту. Для перегляду всіх — треба підтвердити надіслання запиту, після чого відбудеться перенаправлення на окрему сторінку зі знайденими товарами та категоріями.

Головний слайдер розміщений у центрі сторінки. З огляду на те, що зображення центровані та намагаються покрити наданий їм простір — вони можуть обрізатися з боків, проте зберігатимуть свої розміри.

Поряд знаходиться невеликий за наповненням слайдер товарів. Його розміщення не є дуже вдалим. Було б краще, якщо б усі наявні слайдери займали всю ширину та розташовувалися паралельно між собою.

Гарним є позначення акційних пропозицій магазину у вигляді шматка кольорової стрічки з відповідним написом.

Одразу після слайдерів та інфографіки розміщений перелік товарів зі знижкою. Є можливість перемикатися між категоріями.

Нижче — «плитки», що ознайомлюють клієнта із постачальниками магазину та популярними категоріями і дають змогу фільтрувати каталог за відповідним критерієм.

На сайті магазину є секція для розміщення тематичних статей, що можуть бути корисними для власників домашніх улюбленців. Тексти доступні для навігації з головної сторінки. Одразу після записів із блогу розміщена загальна інформація про Інтернет-зоомагазин.

Підвал сайту яскравий та інформативний завдяки використанню іконок, що позначають операторів контактних мобільних телефонів, посилання на профілі в різних соціальних мережах. Також наявні елементи навігації, що фільтрують уміст каталогу за обраною спеціальною пропозицією. Передбачене посилання на сторінку з адресами відділень мережі, що відмічаються за допомогою використання сервісу Google Maps.

Якщо давати оцінку загалом — сторінки є естетично привабливими, проте здаються надто перевантаженими інформацією, що заважає зосередитися. Незвичні дизайнерські рішення порушують усталений переважною більшістю Інтернет-магазинів досвід та ускладнює сприйняття навігації сайтом.

Результатом проведеного аналізу можна вважати сформований користувацький досвід стосовно використання Інтернет-магазинів відповідної тематики, що дає уявлення про обов’язкові основні елементи та виконувані функції.

За спостереженнями, найбільш важливими складовими для веб-зоомагазину виявилися:

- слайдер на головній сторінці для сповіщення користувачів про наявні спеціальні пропозиції та останні новини роботи мережі;

- горизонтальне навігаційне меню, що містить посилання не лише на інформаційні сторінки сайту, а й відображає розділи каталогу;
- надання можливості створення акаунта із максимально простою формою реєстрації;
- якісно реалізований кошик із підтримкою опцій додавання, видалення та зміни кількості товару;
- наявність функціонального рядка контекстного пошуку;
- можливість формування «обраного» переліку товарів;
- використання інфографіки у вигляді іконок для позначення важливих елементів інтерфейсу, категорій каталогу, кількісних переваг та виділення контактної інформації;
- зручна навігація між сторінками веб-додатку: перехід на головну сторінку за допомогою натискання на логотип, автоматична поява кнопки прокручування вгору на «високих» за вмістом сторінках, реалізація відображення «хлібних крихт»;
- відображення переліку постачальників продукції, за якими можна виконувати фільтрацію каталогу;
- використання слайдерів для ознайомлення клієнтів із новинками та популярними товарами;
- механізм зворотного зв’язку у вигляді відгуків, коментарів або рейтингів;
- створення адаптивного дизайну, що забезпечить коректне відображення сторінок сайту незалежно від розміру viewport’у.

1.3 Огляд сучасних підходів до розробки веб-додатків

Однією з сучасних тенденцій у веб-розробці зараз є принцип «mobile-first», обумовлений зростаючими об’ємами мобільного трафіку. Це задало напрям для розвитку веб-галузі на подальші роки. На зміну традиційному підходу до створення сайтів у вигляді багатосторінкових додатків, з’явилися нові — односторінкові та прогресивні.

SPA — веб-додаток, який під час взаємодії із користувачем замість перезавантаження сторінки виконує її динамічну зміну [3].

PWA — веб-додаток, що виконується в браузері та поводить себе як нативний [4, с. 3].

Розробка фреймворків та бібліотек, відповідно, рухаючись у ногу з часом, була переорієнтована, щоб задовольнити зростаючий запит на додатки, створені на базі сучасних підходів.

Ключовою особливістю SPA є завантаження HTML-сторінки з необхідними ресурсами у вигляді CSS та JS файлів лише один раз. Генерація контенту покладена на JavaScript. Із сервером відбувається лише обмін даними.

У свою чергу PWA здатний працювати в офлайн-режимі завдяки завантаженню на пристрій. Він може оновлюватися, причому лише ті складові, що зазнали змін. PWA має можливості розширеної взаємодії з користувачем пристрою завдяки використанню системних сповіщень та push-повідомлень. Загалом він поводить себе як звичайний мобільний додаток.

Роблячи вибір на користь SPA або PWA, вдається отримати переваги та супутні недоліки, що зумовлені особливостями підходу до розробки, що обирається.

Перевагами розробки SPA вважаються швидке завантаження сторінок та майже миттєвий перехід між ними — подібно тому, як це відбувається в додатках для комп’ютера або мобільного телефона. До того ж спостерігається скорочення використання трафіку, бо обмін даними проходить без необхідності передачі ресурсів, що завантажуються лише одного разу [5].

Недоліками SPA є проблеми з SEO, бо не всі пошукові системи, що індексують уміст сайтів, можуть запускати JavaScript. Також отримаємо збільшення навантаження на браузер клієнта, оскільки для відображення контенту треба виконати JS сценарії. У разі наявності проблем із безпекою додаток стає більш вразливим до XSS-атак, важче реалізувати обмеження доступу [5].

У свою чергу серед переваг PWA знаходиться швидке, майже миттєве, завантаження сторінок завдяки використанню кешування. На відміну від SPA може працювати в офлайн-режимі. Також спроможний легко оновлюватися в фоні без втручання користувача. Важлива особливість — завантажується виняткове те, що зазнало змін. PWA виглядає та працює як звичайний мобільний додаток, що не займає багато місця в порівнянні з нативними та не потребує встановлення. На нього навіть можна зробити ярлик на екрані.

У підходу існують також і свої недоліки. У першу чергу — це обмежений доступ до функцій пристрою, якщо порівнювати з нативними додатками. Також маємо підвищення витрат заряду батареї на виконання JavaScript. Нативні мобільні додатки виграють у швидкодії, оскільки оболонкою для виконання PWA виступає браузер. До того ж відсутня можливість публікації в магазині додатків.

Незважаючи на те, що нові підходи отримали широке розповсюдження, продовжує своє існування велика кількість сайтів, побудованих за принципом багатосторінкових додатків (MPA). Таке явище пояснюється неможливістю відмовитися від переваг старого підходу, серед яких виділяють пошукову оптимізацію, масштабованість та безпеку. Також причиною може бути інертність сфери бізнесу.

1.4 Паттерн MVC

Паттерн або шаблон — опис зразка рішення типових для обраної сфері діяльності задач. У залежності від галузі застосування класифікують паттерни аналізу, проектування, програмування, рефакторингу, тестування та інші.

Застосування паттернів проектування є найбільш поширеною практикою. У залежності від ступеня деталізації виокремлюють архітектурні паттерни, шаблони проектування ООП та ідіоми. Перші дають опис структури інформації системи в цілому, визначають у ній підсистеми та їх взаємодію. Другі — у свою чергу діляться ще на три групи:

- твірні — дозволяють абстрагуватися від процесу створення об’єктів класів;

- структурні — дозволяють організовувати більші за розміром структури з класів та їх об’єктів;
- поведінкові — встановлюють способи взаємодії між класами та об’єктами.

Ідіоми, як останній ступінь деталізації, відповідають за вирішення типових проблем із прив’язкою до конкретної мови програмування.

Паттерн MVC є архітектурним паттерном. Він є найбільш популярним вибором під час виконання проектування веб-додатків. Його основні складові визначаються назвою:

- модель (model) — інкапсулює в собі дані та застосовний до них функціонал;
- вид (view) — відповідає за відображення даних із моделі;
- контролер (controller) — виконує обробку дій користувача, пов’язує вид та модель між собою.

1.5 Технологічні підходи до розробки Інтернет-магазинів

Існує декілька технологічних підходів до розробки Інтернет-магазинів. Найбільшою популярністю користуються ті, що базуються на CMS або фреймворках. Також існує варіант ручної розробки безпосередньо на мові програмування.

CMS — веб-додаток, що містить інструменти, які дозволяють додавати, редагувати, видаляти веб-сторінки та їх контент за допомогою браузера та без необхідного володіння знаннями з технологій веб-розробки [6, с. 101].

Прикладами таких систем є: Bitrix, Magento, WordPress та ін. Ці системи являють собою готові шаблони з певним набором функцій для майбутніх Інтернет-магазинів.

CMS може забезпечити базові потреби, тому вони вдало підходять для створення Інтернет-магазину одягу, бутиків, доставок їжі.

Переваги застосування:

- доступність веб-розробки (не треба знати мови програмування);
- швидкість створення сайту та зручне керування;
- структура Інтернет-магазину вже створена;

- реалізований базовий функціонал;
- легка зміна дизайну шляхом встановлення іншого шаблона;
- технічна підтримка від компанії-розробника CMS (для деяких CMS, таких як Bitrix, Magento).

Недоліки:

- відсутність унікальності;
- обмежений функціонал (розширення не завжди можливе або потребує значних витрат);
- надлишковість функціоналу (зумовлена універсальністю);
- повільність завантаження;
- створення унікального дизайну потребує витрат.

Фреймворк — структура, що допомагає в розробці динамічних веб-сайтів, додатків та сервісів [7, с. 109].

Існує велика кількість популярних фреймворків для різних мов програмування. Наприклад, Spring — для Java, Yii — для PHP, Vue.js — для клієнтського JavaScript'у.

Фреймворк не є готовим рішенням. Його призначення — надати програмістові якісні базові складові елементи, що будуть використані для спрощення процесу розробки. Таким чином маємо більше свободи.

Переваги використання фреймворку:

- реалізація будь-яких вимог замовника;
- можливість створення унікального проекту;
- стандартизований підхід до розробки;
- масштабованість із мінімальними втратами швидкодії;
- сайти, що базуються на фреймворках, працюють швидше своїх CMS-аналогів;
- використовується лише потрібний функціонал.

1.6 Огляд систем керування базами даних

Виділяють два основні класи баз даних та відповідних систем, що їх контролюють: SQL (реляційні) та NoSQL (у значенні «not only SQL»).

Реляційна база даних — композиція з однієї або декількох пов’язаних між собою відношеннями таблиць із даними [8, с. 543]. Їх системи керування засновані на однойменній моделі, що була запропонована Едгаром Коддом у 1970 році. Вона складалася з набору об’єктів (або відношень), переліку операторів для взаємодії із відношеннями та засобів підтримки цілісності даних. Загалом вона може бути описана за допомогою 12 правил Кодда. Майже будь-яка задача зі зберігання, читання та отримання даних може бути розв’язана за допомогою реляційних баз даних.

NoSQL — клас СКБД, внутрішня організація яких не підкорюється реляційній теорії та орієнтована на ефективне вирішення окремої проблеми [9, с. 902]. Наприклад: Memcache — нереляційна СКБД із типом «ключ-значення». Вона використовується для тимчасового збереження інформації, що може бути отримана за відомим ключем. Також є документо-орієнтована СКБД MongoDB — вона може бути використана на випадок, коли нам відомо про існування внутрішньої структури всередині документів, що зберігаються. Крім того в класі NoSQL представлені графові СКБД. У якості прикладу — Neo4j.

1.7 Постановка задачі

Мережа зоомагазинів «FanZoo» спеціалізується на продажу товарів для тварин. На поточний момент часу кількість її відділень налічує 5. Вони розташовані у двох містах України: Сумах та Києві. Власник мережі приділяє значну увагу якісному обслуговуванню: передбачена цілодобова підтримка, довгострокова гарантія на акваріуми, спрощена процедура повернення та обміну товару, контроль за якістю наявного асортименту та постійний моніторинг термінів придатності кормів, ласощів та ветеринарних препаратів. Завдяки клієнтоорієнтованому підходу вдається завоювати прихильність відвідувачів магазинів та, як результат, отримати постійних клієнтів. Керівнику важливо отримувати

зворотній зв’язок не лише щодо якості обслуговування, а й разом із тим нові ідеї, що можуть стати корисними для подальшого розвитку бізнесу.

Для підтримки свого функціонування та подальшого розвитку мережа зоомагазинів потребує такий засіб автоматизації, що:

- дасть можливість збільшити кількість клієнтів;
- зможе надавати користувачам актуальну інформацію щодо асортименту та працюючих відділів;
- запропонує відвідувачам зручну навігацію та пошук товарів;
- володітиме функціоналом для оформлення замовлень;
- спростить та пришвидшить обробку замовлень для менеджменту.

У результаті проведеного аналізу існуючих продуктів-аналогів на предмет виявлення основних структурних складових та необхідних користувацьких функцій, а також із врахуванням специфіки ведення бізнесу керівництвом мережі зоомагазинів «FanZoo», був сформований список завдань для реалізації:

- головна сторінка сайту;
- статичні інформаційні сторінки;
- авторизація та профіль користувача;
- перегляд каталогу із можливістю застосування сортування та фільтрації;
- контекстний пошук товарів;
- функціонування кошику та розділу «Обраного»;
- оформлення замовлень;
- адміністрування контенту веб-додатку та обробка замовлень.

За умови успішної реалізації отримаємо веб-магазин, у якому передбачений повний цикл його функціонування: від реєстрації адміністратором відділу мережі в системі, додавання категорій, типів продукції та товарів до створення й обробки замовлення користувача. Очікується, що сайт буде функціонувати без помилок та відображатися однаково у найбільш поширених веб-браузерах, таких, як, Google Chrome, Mozilla Firefox, Microsoft Edge на версіях, датованих 2018 роком та новішими.

2 ВИБІР МЕТОДУ РІШЕННЯ

2.1 Технології для створення серверної частини

На сьогоднішній день існує велика кількість мов програмування, які можна використовувати для створення повноцінної реалізації серверної частини веб-додатку. Здійснення вибору між ними з перебігом часу стає все менш суттєвим, оскільки кожна поступово вдосконалюється, незалежно від часу свого існування.

Це явище можна пояснити тим, що старі мови розробки пройшли довгий шлях свого існування, а в нових творці намагалися усунути недоліки попередників ще на етапі появи.

Різниця може бути лише в синтаксисі та специфіці, що визначає складність розв’язання поставленої задачі. Таким чином, те, що однією мовою реалізується декількома рядками коду, в іншій може потребувати написання більш складних та громіздких конструкцій, що, тим не менше, приведуть до аналогічного результату виконання.

Більш суттєвим, особливо під час розробки великого проекту, є вибір фреймворку, бо їх можливості між собою можуть значно відрізнятися, причому навіть у межах однієї мови програмування.

Обов’язково треба брати до уваги рівень абстракції від предметної області — чим він вищий, тим менше коду. Крім того варто враховувати, із якими елементами можливою буде інтеграція, рівень складності оволодіння технологією. Не менш важливим є розміри спільноти та частість запитів у пошукових системах, що може свідчити про наявність підтримки фреймворку та ймовірності отримання відповідей на питання, що можуть виникати під час розробки.

Якщо спиратися на статистику від statisticsanddata.org, що збиралася майже протягом останніх десяти років, можна з’ясувати, які фреймворки були найбільш затребуваними на той проміжок часу (див. рис. 2.1).

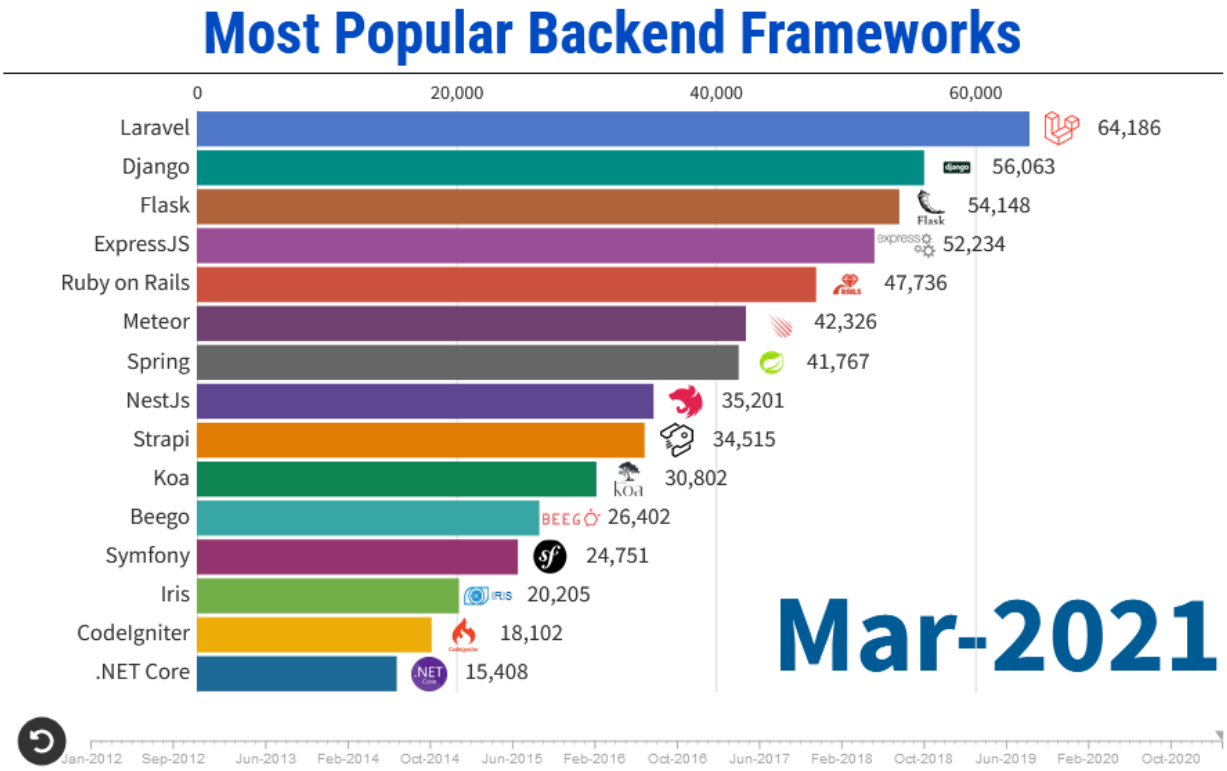

Рисунок 2.1 — Рейтинги фреймворків для розробки серверної частини веб-додатків станом на березень 2021 року [10]

Станом на березень 2021 року лідером серед усіх є Laravel — фреймворк, написаний на мові PHP.

Серед його переваг виділяють:

- artisan — консольний інструмент, що використовується для запуску міграцій, створення користувачів та записів у базі даних вручну, а також для виконання інших разових завдань [11, с. 43];
- підтримка архітектурного патерну MVC, що дозволяє впровадити принцип єдиної відповідальності — кожен окремий клас займається вирішенням однієї окремої задачі;
- вбудований шаблонізатор Blade, за допомогою директив якого зручно будувати види, що повертаються користувачу;
- механізм міграцій дозволяє легко задати структуру таблиць використовуваної бази даних у вигляді коду, що може бути збережений у репозиторії;
- наявність інтеграції з поштовими сервісами для надсилання повідомлень;

- використання ORM — технології об’єктно-реляційного відображення, що дозволяє взаємодіяти із вмістом бази даних за допомогою об’єктів класів;
- надання належного рівня безпеки із захистом від SQL-ін’єкцій та XSS-атак;
- представлений механізм валідації вхідних даних із великою варіативністю правил, що можна комбінувати та доповнювати власними;
- масштабованість;
- пришвидшує розробку.

З огляду на високу популярність Laravel та його переваги було прийнято рішення про використання цього фреймворку для розробки сайту мережі зоомагазинів.

Останні можливості, передбачені в Laravel 9-ї версії, потребують версії PHP не нижче 8.

Для створення початкових та завантаження необхідних файлів будемо використовувати Composer — інструмент керування залежностями для мови PHP [12, с. 2].

2.2 Технології для реалізації клієнтської частини

Для виконання збірки файлів клієнтської частини повинно бути встановлене середовище виконання під назвою Node.js та менеджер пакетів npm.

Відображення контенту сторінок буде спиратися на можливості клієнтських технологій — HTML, CSS та JavaScript. Із метою підвищення зручності під час формування стилів буде використовуватися SASS — CSS-препроцесор, скриптова мова, що розширює синтаксис CSS [13, с. 1]. Його перевагами є:

- можливість використання різного типу змінних, значення яких будуть підставлятися в місця використання в момент формування файлу CSS за допомогою можливостей інтерполяції;
- підтримка вкладеного синтаксису;
- можна генерувати однотипні фрагменти за допомогою циклів;
- використання mixin’ів, що схоже на trait’и в мові PHP, за необхідності можуть бути параметризованими;

- наявні вбудовані функції для роботи з різними типами даних;
- можливість визначення власних функцій;
- наявність способу поєднання декількох фрагментів у один файл стилів за допомогою імпорту;
- можна виконати розрахунок колірних палітр;
- додає можливість визначення однорядкових коментарів.

Варто зазначити, що цей препроцесор підтримує різний синтаксис у залежності від розширення файлу з описом стилів. Якщо SCSS — є необхідність у використанні фігурних дужок та крапки з комою. В іншому випадку, коли це файл SASS — цього робити не можна: препроцесор буде орієнтуватися за розстановкою пробілів.

Для позначення відділів мережі зоомагазинів на карті будуть використовуватися можливості Google Maps API.

2.3 Компонентний підхід під час розробки

Під застосуванням компонентного підходу мається на увазі розбиття програмної реалізації на окремі компоненти із дотриманням принципу єдиної відповідальності — таким, що кожен виконує лише окрему поставлену перед ним задачу.

Зазначений підхід застосовний для розробки як серверної, так і клієнтської частини веб-додатку у рамках використання бібліотек та фреймворків, які в силу своїх внутрішніх особливостей, обумовлених закладеними в їх основі патернами проектування, визначають архітектуру у вигляді розподілу логіки між окремими складовими.

У результаті проведеного аналізу було з’ясовано, що сучасні тенденції розвитку галузі веб-розробки спрямовані на створення додатків, які є максимально схожими на їх мобільні аналоги. Для досягнення цієї мети продовжують свій розвиток такі відомі фреймворки для розробки клієнтської частини, як Vue.js та Angular. Також залишається затребуваною бібліотека React. Утім, з огляду на проблеми з SEO та безпекою, їх застосування для задоволення попиту з боку

сфери бізнесу, що переважно представлена великими багатосторінковими додатками, залишається обмеженим та малоефективним. У свою чергу фреймворки для створення серверної частини не накладають обмежень на тип створюваного додатку.

Було виконано декілька спроб із подолання неуніверсальності сучасних інструментів розробки клієнтської частини веб-додатків, що дозволило сформувати уявлення про проблематику їх застосування стосовно MPA.

Складнощі з SEO пов’язані з відсутністю можливості виконувати JavaScript-сценарії деякими пошуковими роботами. Частим рішенням для подолання цього розглядається застосування SSR — серверного генерації вмісту сторінки, завдяки чому пошукові системи зможуть виконати індексацію контенту. Цей підхід не без недоліків: маємо зростання навантаження на сервер та створення обмежень на реалізацію серверної частини додатку. У такому випадку вона обов’язково повинна спиратися на Node.js під час роботи.

Застосування гідратації — операції, під час якої React намагається прив’язати обробники подій до вже існуючої розмітки [14] — не є вдалим рішенням у разі спроби подолання обмежень на реалізацію серверної частини, оскільки виникає необхідність у постійній підтримці узгодженості коду серверного та клієнтського компонентів.

Також був розглянутий варіант відмовитися від використання фреймворків та перейти на застосування веб-компонентів — набору технологій, що дозволяють створювати елементи з інкапсульованим функціоналом для повторного використання [15]. Утім, їх використання має не менше проблем. Знову ж — це генерація контенту за допомогою JavaScript, що потягне за собою проблеми з SEO. Не менш суттєвою є недостатня підтримка браузерами.

З огляду на те, що метою дипломної роботи є розробка веб-додатку, який буде підтримувати діяльність мережі зоомагазинів, рішення питань, пов’язаних із SEO, є надзвичайно важливим, оскільки під час виходу бізнесу на світовий ринок шляхом публікації сайту в мережі Інтернет застосування механізмів по-

шукової оптимізації дозволяє швидше досягти високих позицій у пошуковій видачі на релевантні запити.

Для реалізації клієнтської частини веб-додатку прийнято рішення створити невеликий JavaScript-фреймворк, що забезпечував би виконання заданої логіки із застосуванням компонентного підходу. Це дозволить створити якісну програмну реалізацію веб-додатку.

Створюваний фреймворк буде базуватися на архітектурному паттерні MVC. До того ж він повинен бути легким, тобто створювати лиш мінімальні накладні витрати. Передбачається застосування механізмів об'єктно-орієнтованого програмування для перевикористання коду, забезпечення подальшої масштабованості та підтримуваності проекту.

Компоненти в цій концепції — це не скільки про класи, що використовуються в межах MVC, а про ті, що підтримуватимуть логіку функціонування окремих фрагментів сторінок сайту, такі, як модальне вікно або корзина.

У фреймворку буде реалізована модульність із дотриманням принципу єдиної відповідальності. Таким чином кожен окремий компонент повинен вирішувати окрему, поставлену перед ним задачу.

Передбачено використання можливостей SSR для вирішення можливих проблем із пошуковим рейтингом, щоб пошукові роботи, які нездатні виконувати JavaScript, могли проіндексувати вміст сайту.

Підтримка життєвого циклу компонентів відбуватиметься максимально автоматизовано та не потребуватиме від розробника, що використовує фреймворк, запуску додаткових команд із управління та налаштування понад тих, що передбачені бізнес-логікою.

Фреймворк не створюватиме обмежень на реалізацію серверної частини веб-додатку. Не менш важливим пріоритетом є підтримуваність більшістю сучасних браузерів. Вимоги до реалізації безпеки та масштабованості, які є серед основних переваг MPA, будуть враховані.

2.4 Проектування інформаційної системи

IDEF0 — це нотація, що використовується для побудови функціональних моделей, які подають у структурованому вигляді інформацію про функції, дії та процеси в рамках системи, що моделюється, або предметної області.

Шляхом використання діаграм, що будуються на основі цієї нотації, можна ретельно дослідити бізнес-процеси, що відбуваються. До того ж здатність до декомпозиції дозволяє це зробити на різних рівнях деталізації.

На IDEF0-діаграмах прийнято позначати діяльність, яка моделюється, у вигляді блоків. Вони приймають потоки:

- зліва — входи або ресурси, які будуть використовуватися;
- справа — виходи або результати виконання активності;
- зверху — керуючі впливи, що визначають кількість та уміст результатів, що будуть отримані;
- знизу — механізми, які описують інструменти, за допомогою яких діяльність буде виконана.

За допомогою програми AllFusion Process Modeler r7 було виконане структурно-функціональне моделювання бізнес-процесів у нотації IDEF0. Контекстна діаграма, що описує основну активність «Робота Інтернет-зоомагазину “FanZoo”» в узагальненому вигляді, зображена на рис. 2.2. Виявили, що ресурсами цього процесу є дані про користувача, інвентаризацію та уміст каталогу товарів. Вони перетворюються на виходи, серед яких можуть бути: оброблене замовлення, оновлений уміст каталогу та інвентаризації, зібрані аналітичні дані. Таке перетворення виконується за рахунок дій користувача, адміністратора та функціонування з використанням наявних апаратних можливостей розробленої програмної реалізації. Виконання бізнес-процесу регулюється законодавством, вимогами замовника програмного продукту. Усе інше, що не регламентується документами, спирається на найкращі практики. Ними можемо вважати негласні правила, сформовані на користувацькому досвіді.

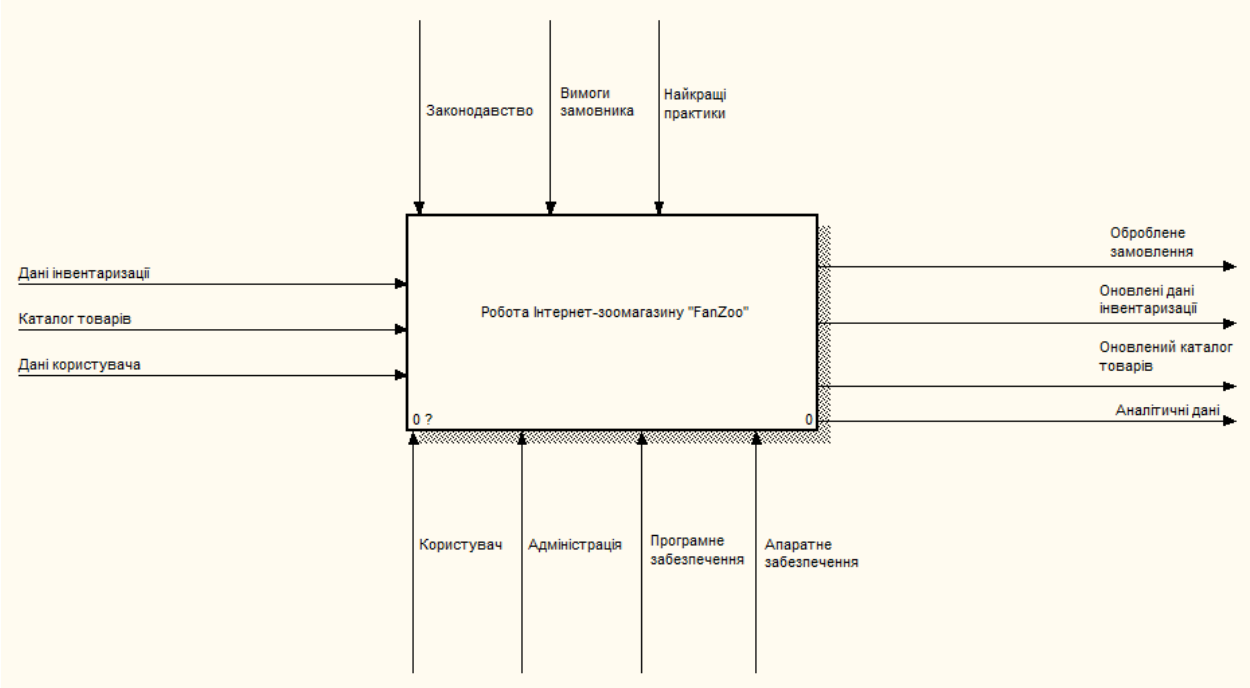

Рисунок 2.2 — Контекстна діаграма бізнес-процесу «Робота Інтернет-зоомагазину “FanZoo”» у нотації IDEF0

Результати виконання декомпозиції основного бізнес-процесу представлені на рис. 2.3.

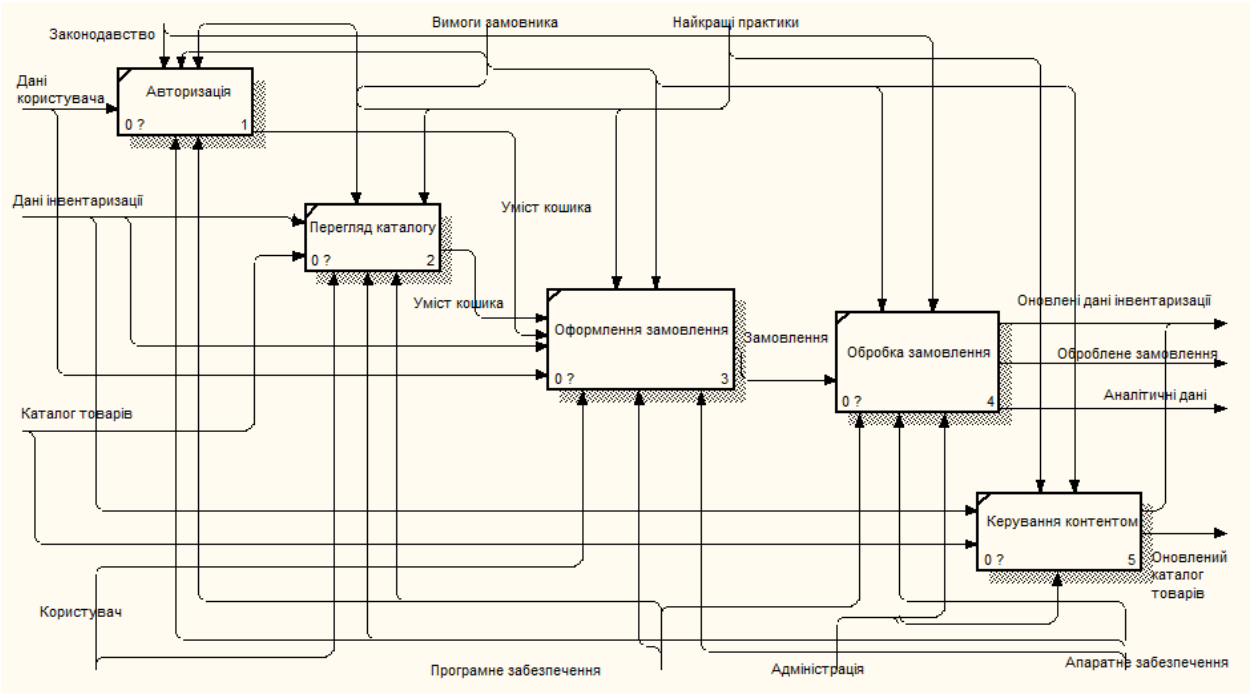

Рисунок 2.3 — Діаграма декомпозиції бізнес-процесу «Робота Інтернет-зоомагазину “FanZoo”» у нотації IDEF0

Для більш детального опису того, що відбувається в основній діяльності, знадобилося зобразити 5 блоків, що позначають підпроцеси. Вони відповідають за найголовніші аспекти функціонування веб-додатку, такі, як авторизація, перегляд каталогу, оформлення замовлення, обробка замовлення та керування контентом. Можемо бачити, що всі потоки, визначені раніше, на рівні контекстної діаграми, розподілені між підпроцесами. Також під час роботи додатку відбувається створення нових — проміжних вхідних ресурсів, що передаються як результат від однієї активності на обробку до іншої. Наприклад: уміст кошика після перегляду каталогу буде отриманий у якості вхідного ресурсу підпроцесом, що займається створенням замовлень.

Варіант використання — частина функціоналу системи, застосовувана користувачами з метою отримання результату.

Актор — категорія користувачів програмного продукту, що застосовують доступну їм функціональність системи.

Діаграма варіантів використання — графічний спосіб подання набору акторів, варіантів використання та зв’язків між ними.

Існують два типи зв’язків: include та extend. Перші вказують на варіанти використання, що будуть обов’язково використані в поточному. Другі — на ті, перехід до застосування яких можливий лише за виконання певних умов.

У результаті опрацювання наявних відомостей про майбутній веб-додаток вдалося виділити акторів (див. табл. 2.1) та пов’язані з ними варіанти використання (див. табл. 2.2).

Таблиця 2.1 — Актори

Актор	**Опис**
Неавторизований користувач	Гість веб-сайту, має доступ до інформаційних сторінок, може створювати замовлення
Користувач	Авторизований користувач, має доступ до сторінок профілю, володіє додатковими можливостями в порівняні з неавторизованим

Продовження таблиці 2.1

Актор	**Опис**
Адміністратор	Здійснює обробку замовлень користувачів та керування вебдодатком шляхом встановлення значень параметрів, редагування вмісту каталогу
Google Maps API	Зовнішній сервіс, що використовується додатком для відображення положень магазинів на карті

Таблиця 2.2 — Варіанти використання

Варіант використання	**Опис**
Реєстрація	Створення нового акаунту
Авторизація	Вхід до існуючого акаунту
Вихід	Вихід із акаунту, якщо була виконана авторизація
Надіслання відгуку	Надіслання повідомлення для підтримки зворотного зв’язку від авторизованого користувача
Перегляд профілю	Перегляд сторінок профілю авторизованим користувачем
Редагування персональних даних	Оновлення користувачем інформації про себе
Зміна пароля	Встановлення нового пароля, що використовується для входу в акаунт у разі правильного вказання попереднього
Перегляд замовлень	Перегляд актуальної інформації щодо оформлених раніше замовлень
Перегляд інформаційних сторінок	Наявність доступу до сторінок зі статичним умістом
Перегляд каталогу	Ознайомлення зі вмістом каталогу
Пошук	Застосування контекстного пошуку для отримання списку релевантних товарів
Фільтрація каталогу	Можливість визначення параметрів для зменшення кількості отримуваних результатів перегляду каталогу
Дії над кошиком	Додавання, зміна кількості та видалення товарів
Дії над обраним	Додавання та видалення товарів

Продовження таблиці 2.2

Варіант використання	**Опис**
Створення замовлення	Оформлення замовлення
Обробка замовлень	Адміністратор може оновлювати статуси оформлених у веб-додатку замовлень
Налаштування сайту	Вказання кількісних значень параметрів: розмір «новинок», «хітів продажів», та інших
Перегляд зворотного зв’язку	Перегляд переліку повідомлень від зареєстрованих користувачів
Керування контентом	Редагування вмісту каталогу та вмісту веб-додатку в цілому

Для створюваного проекту побудова діаграми варіантів використання була виконана за допомогою програми Adobe Photoshop (див. рис. 2.4).

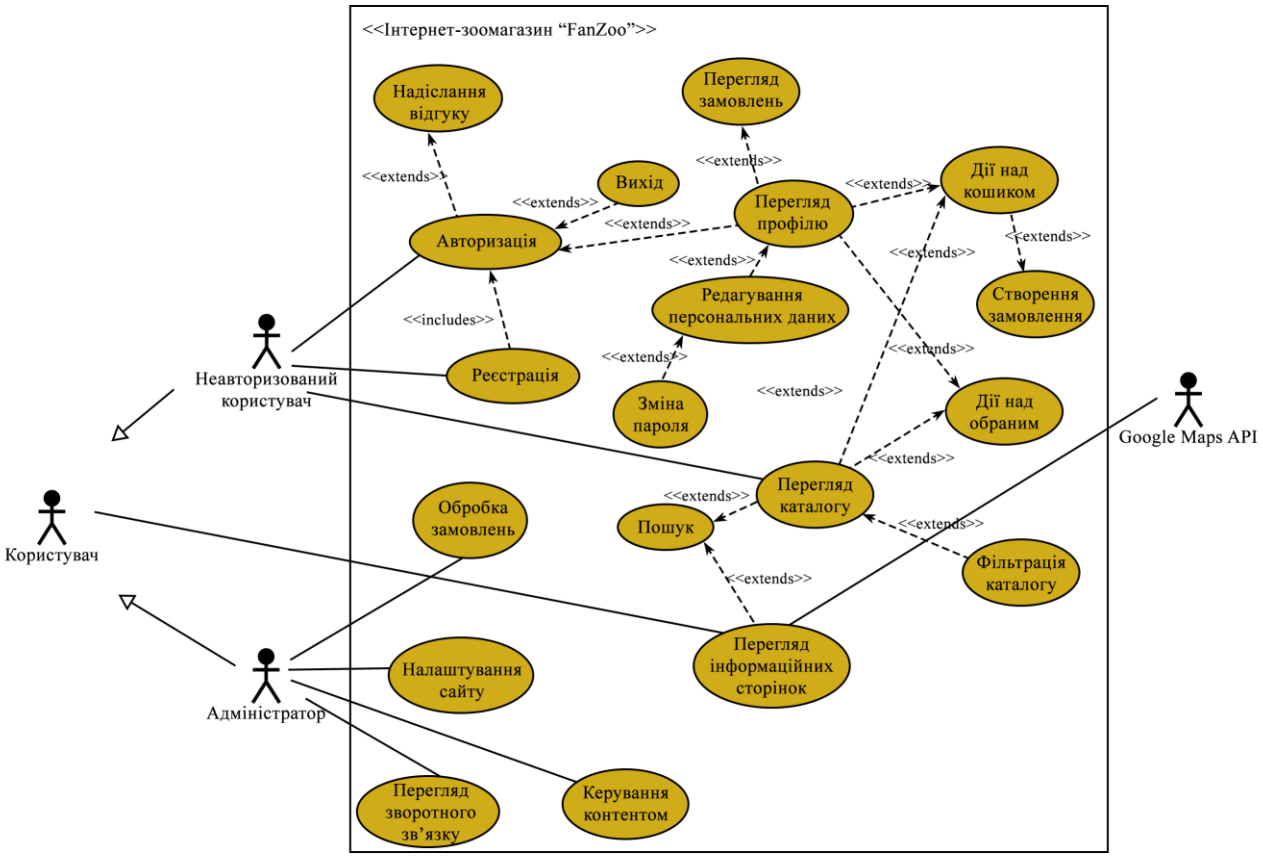

Рисунок 2.4 — Діаграма варіантів використання веб-додатку

Усі згадані раніше актори та варіанти використання були позначені.

2.5 Проектування моделі бази даних

Перш за все визначимося із класом СКБД, що будемо використовувати. Обираємо реляційні, оскільки будемо працювати зі структурованими даними. До того ж, оскільки створюваний додаток спрямований на підтримку діяльності бізнесу, то надзвичайно важливим є підтримка цілісності даних.

Поширеним вибором серед реляційних СКБД для веб-додатків є MySQL та MariaDB — його відгалуження. Надаємо перевагу останньому з огляду на впроваджені чисельні оптимізаційні механізми в порівнянні з оригінальним MySQL.

Модель бази даних може бути представлена за допомогою ER-діаграми — графічного способу представлення структури таблиць у вигляді сутностей та зв’язків між ними.

Виділяють три типи ER-діаграм у залежності від рівня деталізації: концептуальна, логічна та фізична.

Концептуальна надає інформацію про те, які сутності визначені в системі та які зв’язки між ними встановлені. Вона не дозволяє точно визначити всі необхідні таблиці, що повинні бути створені в базі даних для підтримки затребуваних бізнес-правил. Наприклад, зв’язок між двома сутностями потужністю M до N, який ще має назву «багато до багатьох», на концептуальній ER-діаграмі не відображатиме необхідності передбачити проміжну сутність, для котрої у базі даних повинна бути створена ще одна таблиця.

Логічна — відповідає за наступний рівень деталізації ER-діаграми. У порівнянні з концептуальною додається вказання атрибутів, яким будуть відповідати колонки в реляційній моделі. Використання типізації є необов’язковим. На цьому рівні припускається збереження незалежності від постачальника бази даних, тобто прив’язка до специфіки реалізації відсутня подібно тому, як це було на попередньому рівні.

Фізична — фактично є планом структури таблиць бази даних з огляду на свою деталізованість, що бере до уваги обмеження, які застосовуються щодо

атрибутів сутностей, такі як: тип, довжина значення, допустимість значення null, унікальність, також вказання первинних та зовнішніх ключів. На цьому рівні враховуються особливості обраної для використання бази даних, що може стати причиною жорсткої прив’язки до конкретної реалізації.

З метою отримання максимально корисних результатів проектування для створення веб-додатку було вирішено зупинитися на фізичному рівні деталізації ER-діаграми. Наявність конкретної інформації не лише щодо сутностей та зв’язків між ними, а й про їх уміст дозволить прискорити розробку.

MySQL Workbench — програмне забезпечення, що було обрано для використання під час проектування схеми таблиць бази даних.

З огляду на те, що проект СКБД MariaDB є відгалуженням від MySQL, то раніше згаданий інструмент може бути використаний для проектування, бо маємо ідентичну реляційну модель та аналогічні типи даних для колонок таблиць.

Оскільки очікується використання Laravel, усередині якого для роботи з базою даних використовується технологія об’єктно-реляційного відображення, що в рамках фреймворку має назву «Eloquent» (від англ. «промовистий») в офіційній документації, тому, з метою зменшення об’ємів коду необхідно дотримуватися конвенцій іменування, що використовуються в межах фреймворку, а саме: назва таблиця — це ім’я відповідної сутності (або моделі, якщо використовувати термінологію, передбачену паттерном MVC) у множині. Проміжні таблиці, що відповідають за реалізацію відношень «багато до багатьох» отримують назви відповідних моделей в однині, з’єднані нижнім підкресленням. Є опція виокремлення в окрему модель із власною назвою. Також Eloquent очікує, що буде визначений стовпчик під назвою «id» у якості первинного ключа в кожній таблиці. Відповідно, іменування поширюється на зовнішні ключі: очікуваною назвою за замовчуванням є назва зовнішньої моделі в однині, що супроводжується нижнім підкресленням та «id».

Незважаючи на те, що описана поведінка може бути переналаштована — явно прописати назву ключа, змінити тип даних або відключити автоінкремент

для первинного ключа, якщо той не є цілим числом, також існують особливості, що треба обов’язково враховувати. Однією з них є відсутність підтримки використання складених первинних ключів, тому під час формування діаграми ідентифікуючі зв’язки — такі, за використання яких відбувається становлення зовнішнього ключа частиною первинного — не можуть бути використані.

Із метою спрощення роботи з датою та часом для Eloquent поведінкою за замовчуванням є додавання міток часу у вигляді полів «created_at» та «updated_at». Їх значення визначаються автоматично — у момент створення або оновлення запису в базі даних відповідно. Цю опцію можна відключити, але досить часто її наявність є необхідною — наприклад, для відслідковування мітки часу створення акаунта користувача або визначення моменту оновлення його особистих даних. Також у фреймворку врахований варіант, коли необхідним є лише один із цих атрибутів. Для зручності використання структури таблиць, що розробляється, буде дотримане стандартне іменування цих полів.

Eloquent також пропонує можливість так званого «м’якого видалення», де основна ідея — використання додаткового атрибуту «deleted_at», значенням якого може бути null або мітка часу. У другому випадку — рядок таблиці буде вважатися видаленим, але при цьому не покине базу даних, що дозволить зберегти пов’язані з ним записи.

Створена фізична ER-діаграма представлена на рис. 2.5.

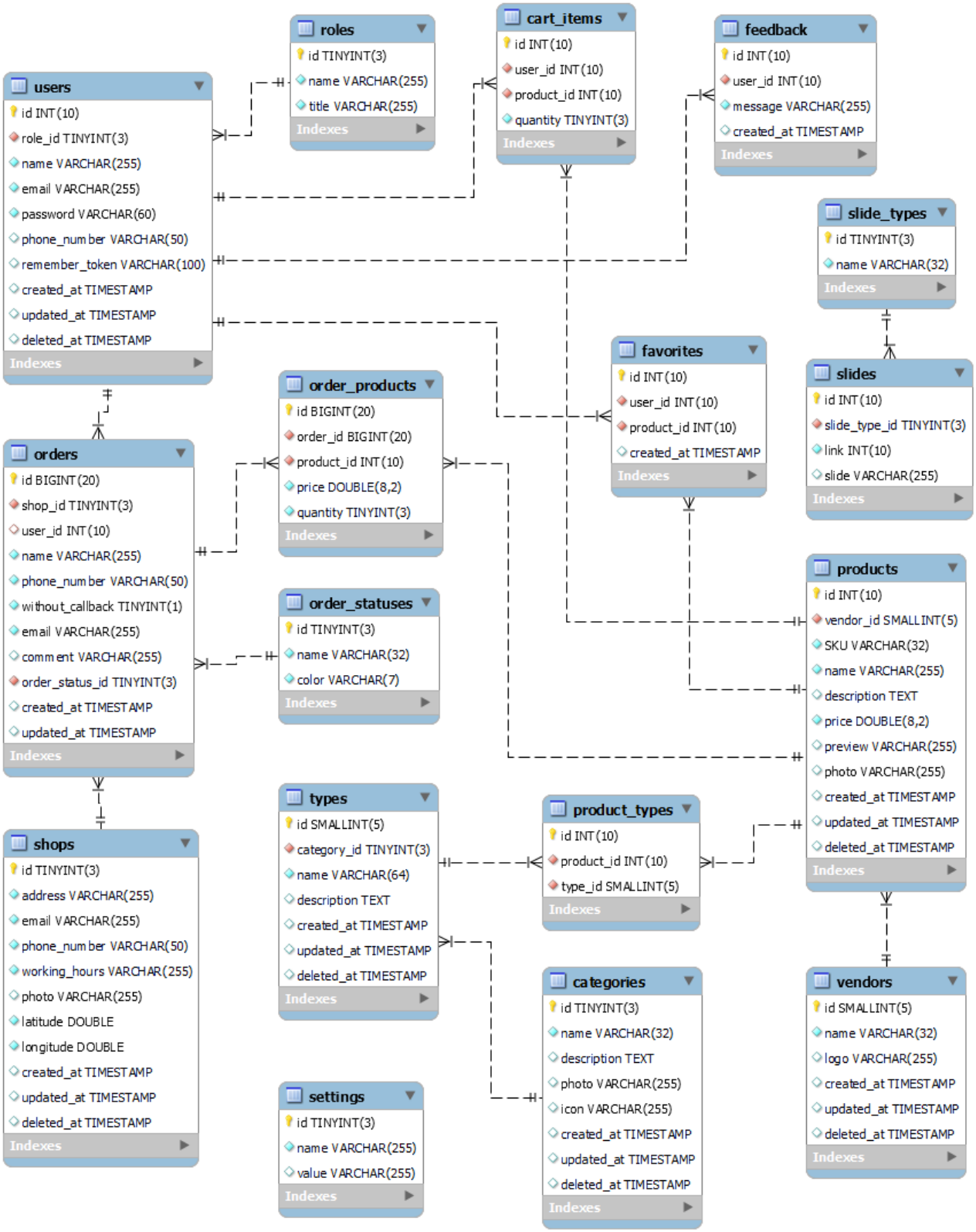

Рисунок 2.5 — Фізична ER-діаграма бази даних веб-додатку

Усі атрибути сутностей є функціонально залежними від визначеного в кожній із них первинного ключа. Назви таблиць відповідають тим, що обрані для іменування сутностей. Розглянемо кожну сформовану сутність окремо, щоб

з’ясувати, за що відповідають передбачені атрибути, які обмеження накладаються на їх значення, а також наявні особливості використання веб-додатком.

«roles» описує можливі в системі ролі користувачів (див. табл. 2.3).

Таблиця 2.3 — Атрибути сутності «roles»

Атрибут	**Опис**	**Ключовий**	**Обмеження**
id	Ідентифікатор ролі	PK	Непорожній, невід’ємний
name	Внутрішня назва ролі		Непорожній, унікальний
title	Назва ролі для відображення		Непорожній, унікальний

Значення атрибутів «name» і «title» повинні бути окремо унікальними та непорожніми.

«users» зберігає інформацію про зареєстрованих у системі користувачів (див. табл. 2.4).

Таблиця 2.4 — Атрибути сутності «users»

Атрибут	**Опис**	**Ключовий**	**Обмеження**
id	Ідентифікатор користувача	PK	Непорожній, невід’ємний
role_id	Ідентифікатор ролі	FK — «roles»	Непорожній, невід’ємний
name	Прізвище та ім’я		Непорожній
email	Електронна пошта		Непорожній, унікальний
phone_number	Номер телефону		
remember_token	Токен для опції «Запам’ятати мене»		
created_at	Момент створення		
updated_at	Момент оновлення		
deleted_at	Момент видалення		

Атрибут «name» може зберігати і ім’я, і прізвище, або будь-яке інше звернення, яке вибере для себе користувач. На рівні бази даних та додатку воно бу-

де сприйматися як атомарне значення, тобто відсутні ділянки коду, де б значення атрибуту розбивалося на окремі складові. «email» повинен бути унікальним. Номер телефону вказувати не обов’язково. Пароль в атрибуті «password» зберігатиметься в зашифрованому вигляді. Номер телефону можна вказати для зручності заповнення форм. Значення «remember_token» генеруватиметься вебдодатком у разі використання опції «Запам’ятати мене», що можна активувати під час проходження авторизації. За замовчуванням фреймворк Laravel не визначає обмежень на значення полів «created_at», «updated_at». У «deleted_at» може вказуватися момент часу видалення запису про користувача або null.

У «categories» знаходяться дані про категорії товарів (див. табл. 2.5).

Таблиця 2.5 — Атрибути сутності «categories»

Атрибут	**Опис**	**Ключовий**	**Обмеження**
id	Ідентифікатор категорії	PK	Непорожній, невід’ємний
name	Назва		Непорожній, унікальний
description	Опис		
photo	Назва файлу фотографії		
icon	Назва файлу іконки (для інфографіки)		
created_at	Момент створення		
updated_at	Момент оновлення		
deleted_at	Момент видалення		

У якості значень «photo» та «icon» очікуються назви файлів із розширенням. Вони будуть знаходитися в папці відповідної категорії, що дозволить зберігати в базі даних максимально короткі відносні стосовно неї шляхи. Виходячи з обмежень помітно, що фотографії можуть бути не завантажені — у такому випадку веб-додаток буде використовувати замість них заповнювачі-замінники. Також будуть відслідковуватися моменти створення, оновлення та видалення категорій.

«types» містить інформації про типи товарів, що прив’язані до відповідних категорій (див. табл. 2.6).

Таблиця 2.6 — Атрибути сутності «types»

Атрибут	**Опис**	**Ключовий**	**Обмеження**
id	Ідентифікатор типу	PK	Непорожній, невід’ємний
category_id	Ідентифікатор категорії	FK — «categories»	Непорожній, невід’ємний
name	Назва		Непорожній
description	Опис		
created_at	Момент створення		
updated_at	Момент оновлення		

«name» не є унікальним, але повинен бути таким у своїй категорії.

«vendors» містить інформацію про постачальників продукції для мережі магазинів (див. табл. 2.7).

Таблиця 2.7 — Атрибути сутності «vendors»

Атрибут	**Опис**	**Ключовий**	**Обмеження**
id	Ідентифікатор постачальника	PK	Непорожній, невід’ємний
name	Назва		Непорожній, унікальний
logo	Назва файлу логотипу		
created_at	Момент створення		
updated_at	Момент оновлення		
deleted_at	Момент видалення		

Логотип, якщо відсутній, буде заміщений за допомогою заповнювача.

Усередині «products» описаний асортимент товарів, яким володіє мережа (див. табл. 2.8). До «name» не висувається умов унікальності, а лише до артикулу. Очікується, що значення «preview» буде формуватися автоматично фрейм-

ворком у разі вказання фотографії продукції. Для зручності адміністратора вказувати постачальника не є обов’язковим.

Таблиця 2.8 — Атрибути сутності «products»

Атрибут	**Опис**	**Ключовий**	**Обмеження**
id	Ідентифікатор товару	PK	Непорожній, невід’ємний
vendor_id	Ідентифікатор постачальника	FK — «vendors»	Непорожній, невід’ємний
SKU	Артикул		Непорожній
name	Назва		Непорожній
description	Опис		
price	Вартість		Непорожній, невід’ємний
preview	Назва файлу попереднього перегляду		
photo	Назва файлу фотографії		
created_at	Момент створення		
updated_at	Момент оновлення		
deleted_at	Момент видалення		

У «product_types» здійснюється прив’язка типів до товару (див. табл. 2.9). Таким чином одна окрема продукція може належати одразу до декількох типів. Наприклад: корм для котів і собак. Для запобігання дублюванню така можливість була передбачена.

Таблиця 2.9 — Атрибути сутності «product_types»

Атрибут	**Опис**	**Ключовий**	**Обмеження**
id	Ідентифікатор запису «продукт-тип»	PK	Непорожній, невід’ємний
product_id	Ідентифікатор продукту	FK — «types»	Непорожній, невід’ємний
type_id	Ідентифікатор типу	FK — «products»	Непорожній, невід’ємний

«shops» містить інформації про відділення мережі (див. табл. 2.10).

Таблиця 2.10 — Атрибути сутності «shops»

Атрибут	**Опис**	**Ключовий**	**Обмеження**
id	Ідентифікатор відділення магазину	PK	Непорожній, невід'ємний
address	Адреса розташування		Непорожній, унікальний
email	Контактна електронна пошта		Непорожній
phone_number	Контактний номер телефона		Непорожній
working_hours	Робочі години		Непорожній
photo	Назва фотографії магазину		
latitude	Широта: від -90 до 90 (Google Maps)		Непорожній
longitude	Довгота: він -180 до 180 (Google Maps)		Непорожній
created_at	Момент створення		
updated_at	Момент оновлення		
deleted_at	Момент видалення		

Встановлення фотографії не є обов'язковим. Значення широти та довготи для зберігання треба визначати вручну за допомогою сервісу Google Maps.

«favorites» — зберігає вміст розділу «Обране» (див. табл. 2.11).

Таблиця 2.11 — Атрибути сутності «favorites»

Атрибут	**Опис**	**Ключовий**	**Обмеження**
id	Ідентифікатор запису про товар із «Обраного»	PK	Непорожній, невід'ємний
user_id	Ідентифікатор користувача	FK — «users»	Непорожній, невід'ємний
product_id	Ідентифікатор товару	FK — «products»	Непорожній, невід'ємний
created_at	Момент створення		

Момент створення «created_at» враховується для сортування — останній доданий до «Обраного» товар повинен відображатися першим.

«cart_items» містить уміст корзини користувача, тим самим дозволяє організувати міжсесійний доступ до сформованої вибірки товарів у кошику (див. табл. 2.12).

Таблиця 2.12 — Атрибути сутності «cart_items»

Атрибут	**Опис**	**Ключовий**	**Обмеження**
id	Ідентифікатор запису товару	PK	Непорожній, невід'ємний
user_id	Ідентифікатор користувача	FK — «users»	Непорожній, невід'ємний
product_id	Ідентифікатор продукту	FK — «products»	Непорожній, невід'ємний
quantity	Кількість одиниць товару		Непорожній, невід'ємний

Пара «user_id» та «product_id» повинна бути унікальною.

«order_statuses» визначає доступні стани обробки замовлень (див. табл. 2.13).

Таблиця 2.13 — Атрибути сутності «order_statuses»

Атрибут	**Опис**	**Ключовий**	**Обмеження**
id	Ідентифікатор статусу обробки замовлення	PK	Непорожній, невід'ємний
name	Назва		Непорожній, унікальний

«orders» містить інформацію про існуючі в системі замовлення та їх поточний стан обробки (див. табл. 2.14).

Таблиця 2.14 — Атрибути сутності «orders»

Атрибут	**Опис**	**Ключовий**	**Обмеження**
id	Ідентифікатор замовлення	PK	Непорожній, невід’ємний
shop_id	Ідентифікатор відділення магазину	FK — «shops»	Непорожній, невід’ємний
user_id	Ідентифікатор користувача	FK — «users»	Невід’ємний
name	Ім’я та прізвище		Непорожній
phone_number	Номер телефона клієнта		Непорожній
without_call	Чи можна дзвонити		Непорожній
email	Електронна пошта		Непорожній
comment	Коментар		
order_status_id	Поточний статус обробки	FK — «order_statuses»	Непорожній, невід’ємний
created_at	Момент створення		
updated_at	Момент оновлення		

Користувач може додати коментар до замовлення та згоду на дзвінок.

В «order_products» зберігається інформація про окремі складові кожного замовлення, а саме: кількість одиниць за кожною позицією та ціна товару на момент оформлення замовлення (див. табл. 2.15).

Таблиця 2.15 — Атрибути сутності «order_products»

Атрибут	**Опис**	**Ключовий**	**Обмеження**
id	Ідентифікатор складової замовлення	PK	Непорожній, невід’ємний
order_id	Ідентифікатор замовлення	FK — «orders»	Непорожній, невід’ємний
product_id	Ідентифікатор товару	FK — «products»	Непорожній, невід’ємний
price	Вартість на момент оформлення		Непорожній, невід’ємний
quantity	Кількість одиниць товару		Непорожній, невід’ємний

Пара значень атрибутів «order_id» та «product_id» повинна бути унікальною. Вартість товару фіксується, щоб дані старих замовлень не могли змінюватися в залежності від поточної вартості продукції. До того ж таким чином легше запровадити механізм нарахування знижок.

«slide_types» визначає типи слайдів, що підтримуються веб-додатком (див. табл. 2.16).

Таблиця 2.16 — Атрибути сутності «slide_types»

Атрибут	**Опис**	**Ключовий**	**Обмеження**
id	Ідентифікатор типу слайду	PK	Непорожній, невід’ємний
name	Назва		Непорожній, унікальний

У свою чергу «slides» зберігає відомості про слайди, що будуть використовуватися на головній сторінці (див. табл. 2.17).

Таблиця 2.17 — Атрибути сутності «slides»

Атрибут	**Опис**	**Ключовий**	**Обмеження**
id	Ідентифікатор слайду	PK	Непорожній, невід’ємний
slide_type_id	Ідентифікатор типу слайду	FK — «slide_types»	Непорожній, невід’ємний
link	Ідентифікатор ресурсу, що рекламується		Непорожній, невід’ємний
slide	Назва файлу зображення слайду		

Значення атрибуту «link» інтерпретується в залежності від типу слайду, тому зовнішній ключ не використовується, що дозволяє уникнути ускладнень структури та надає розширюваність. У той же час варто зазначити, що контроль цілісності даних у такому випадку покладається на веб-додаток. Додавання зображення «slide» є опціональним. У разі його відсутності — слайд не буде відображений.

На сутність «feedback» покладається задача зі збереження користувацьких відгуків (див. табл. 2.18).

Таблиця 2.18 — Атрибути сутності «feedback»

Атрибут	**Опис**	**Ключовий**	**Обмеження**
id	Ідентифікатор відгуку	PK	Непорожній, невід'ємний
user_id	Ідентифікатор користувача	FK — «users»	Непорожній, невід'ємний
message	Текст відгуку		Непорожній
created_at	Момент створення		

Значення атрибуту «user_id» повинно бути обов'язково вказано, оскільки відгуки можуть залишати лише зареєстровані користувачі. «created_at» необхідний для сортування, щоб адміністратор міг переглянути останні в першу чергу.

У «settings» будуть зберігатися налаштування сайту (див. табл. 2.19).

Таблиця 2.19 — Атрибути сутності «settings»

Атрибут	**Опис**	**Ключовий**	**Обмеження**
id	Ідентифікатор	PK	Непорожній
name	Назва налаштування		Непорожній
value	Значення		Непорожній

Варто зазначити, що атрибут «name» є неунікальним, щоб мати можливість, на випадок такої необхідності, змогу зберігати масиви значень.

З метою спрощення роботи з базою даних для всіх передбачених на етапі проектування первинних ключів застосовується автоінкремент і unsigned-обмеження. Це дає змогу створювати послідовності ідентифікаторів, починаючи від 1 в автоматичному режимі.

2.6 Інструменти для розробки

Під час створення Інтернет-магазину, окрім раніше зазначених технологій, знадобиться велика кількість інструментів.

У першу чергу — це інтегроване середовище розробки PhpStorm від компанії JetBrains. Серед її можливостей можна виділити основні:

- форматування написаного коду із врахуванням налаштувань стилю;
- інтелектуальне автодоповнення, що враховує контекст;
- наявність спливаючих підказок та параметрів;
- статичний аналіз якості програмного коду;
- ергономічне середовище за рахунок ефективної навігації та швидкого пошуку;
- підтримка автоматичної компіляції SASS у CSS;
- наявність вбудованих інструментів для проведення безпечного рефакторингу коду;
- має велику кількість якісних плагінів, що розширюють функціонал IDE.

Під час розробки веб-сайту неможливо обійтися без налагоджування та тестування. Для цього виникає потреба в запуску локального веб-сервера. PhpStorm, звісно, має таку можливість, проте більш зручним є використання програмного пакету XAMPP — кросплатформової збірки, що має вбудовані продукти, такі як інтерпретатор PHP, веб-сервер Apache, інструмент phpMyAdmin для адміністрування СКБД MariaDB, що також постачається разом у комплекті. Після встановлення все буде налаштованим та готовим до роботи.

Не менш важливим інструментом є Webpack — збиральник модулів. Основна ідея його функціонування — побудова графу залежностей, починаючи із вказаної точки входу, використовуючи який буде відбуватися процес отримання файлів у правильному порядку для формування збірки, використання яких у проекті має свої переваги, серед яких: зменшення кількості запитів до сервера за рахунок кешування, автоматичне розв’язання залежностей.

За замовчуванням Webpack працює із файлами, що мають розширення js або json, але його можливості можуть бути розширені за рахунок використання завантажувачів. Також може виконувати додаткові дії у разі встановлення плагінів.

Варто додати, що зазначений інструмент постачається неявно разом із фреймворком Laravel, оскільки лежить в основі Laravel Mix.

Для виконання налагодження проекту під управлінням фреймворку Laravel корисним є плагін Laravel Debugbar. Серед його можливостей:

- відображення інформації про запити: їх кількість та час виконання;
- виявлення дублікатів серед запитів, або таких, що не використовують жадібне завантаження;
- демонстрація актуального вмісту сесії;
- наведення деталізованого опису вмісту запиту: шлях, статус, заголовки, параметри різного типу і т. д.;
- формування переліку відображень, які були задіяні під час побудови відповіді сервера;
- обчислення кількості використаних моделей різних типів;
- демонстрація інформації про поточний маршрут.

Для проведення налагодження та тестування необхідним є наповнення бази тестовими даними. Для цього будемо використовувати Faker — бібліотеку, що постачається разом із Laravel. Основними є її можливостями є:

- створення текстів: реалістичних імен та прізвищ, адрес електронної пошти та телефонних номерів, речень та параграфів;
- виконання ролі генератора псевдовипадкових чисел, окремих цифр та логічних значень;
- генерація зображень у вигляді текстових написів на тлі випадкового кольору;
- велика кількість допоміжних функцій, що дозволяють отримувати значення із різних категорій: час і дата, назви вулиць, міст, країн та інші

- передбачена можливість досягти унікального значення за певну максимальну кількість ітерацій;
- формування наповнення, яке задовольняє вказаний шаблон або регулярний вираз;
- гнучка конфігурація.

Також планується до застосування веб-сервіс RealFaviconGenerator, що доступний за посиланням: https://realfavicongenerator.net. Цей інструмент уміє генерувати іконки сайту на основі вказаного зображення. Він є корисним, оскільки:

- надається зручний інтерфейс для налаштування параметрів іконки під різні пристрої та браузери;
- автоматизація процесу створення зображень необхідних розмірів та конфігураційних файлів;
- дозволяє отримати однакового відображення іконки на різних пристроях та браузерах.

Матеріали роботи були представлені на конференції IMA-2022 [16].

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Реалізація клієнтської частини

Фреймворк, що буде підтримувати виконання логіки на клієнтській частині веб-додатку, має таку структуру (див. рис. 3.1)

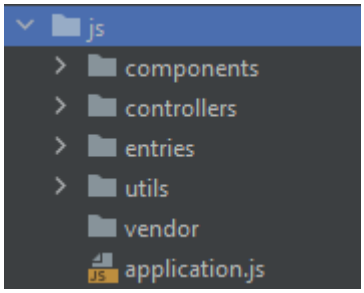

Рисунок 3.1 — Структура файлів JavaScript-фреймворку

Перші 4 з них є основними: components, controllers, entries, utils. У разі виникнення необхідності використання сторонніх розробок передбачено можливість створення додаткової папки vendor. Розглянемо призначення основних папок.

components — містять класи, що інкапсулюють логіку функціонування окремих складових веб-сторінок, що після свого опису у вигляді JavaScript класу проходять реєстрацію. Для імплементації внутрішньої логіки взаємодії можуть містити в середині себе елементи — посилання на об’єкти, створювані браузером для HTML-елементів, та підкомпоненти — їх об’єкти-обгортки. Інколи компоненти можуть надавати публічний доступ до обробників подій — функцій зворотного виклику, які визначаються контролером за необхідності опису взаємодії компонента із зовнішнім середовищем. Класи імпортуються в скриптах із описами контролерів.

controllers — файли класів контролерів, що містять логіку взаємодії компонентів між собою на окремих сторінках. Після визначення проходять реєстрацію в додатку. Імпортуються в точках входу.

entries — точки входу в додаток. Є необхідними для збиральників. Наприклад, WebPack — він буде збирати bundle-файли, використовуючи вказані через

import залежності. Із міркувань безпеки точок входу може бути декілька, що дозволяє розмежувати логіку виконання додатку. Як приклад: для користувача та адміністратора. Створення окремих bundle-файлів дасть можливість приховати приватний API панелі адміністратора від можливих атак.

utils — допоміжні класи, такі як: DOMUtils, ObjectUtils та інші із можливістю виклику корисних функцій без необхідності інстанціювання, бо методи повинні бути об’явлені статичними.

Файл application.js містить однойменний клас Application із основною логікою роботи із зареєстрованими типами компонентів, їх екземплярами та контролерами. Запуск ініціалізації клієнтської частини веб-додатку покладається на кожну окрему точку входу, описану в entries.

Стан клієнтської частини веб-додатку описується приватними статичними полями:

```
static #context = {
  controllers: new Map(),
  components: new Map(),
  componentInstances: new Map(),
};
static #initialized = false;
```

Контекст використовується для реєстрації контролерів, компонентів та їх екземплярів за допомогою відповідних статичних методів:

```
static registerController(controllerClass) {
  this.#context.controllers.set(controllerClass.controllerName,
controllerClass);
}

static registerComponent(componentClass) {
  const componentName = componentClass.componentName;
  this.#context.components.set(componentName, componentClass);
  this.#context.componentInstances.set(componentName, new Set());
}

static registerComponentInstance(component) {
  let prototype = component.constructor;
  while (prototype !== Component) {

this.#context.componentInstances.get(prototype.componentName).add(component);
    prototype = Object.getPrototypeOf(prototype);
  }
}
```

Варто зазначити, що в якості параметрів у перших двох використовуються не рядки, а класи. Це є необхідним для того, щоб отримати назви, що зберігаються в статичних властивостях controllerName, componentName відповідно.

Варто зазначити, що використати name як назву класу — неможна, оскільки його використання немає сенсу в разу застосування мініфікації JavaScript-коду.

У свою чергу додавання екземпляру компонента до контексту має свої особливості. Можемо бачити циклічний пошук усіх прототипів, що повинен зупинитися на суперкласі Component. Таким чином відбувається реєстрація одного й того ж самого об’єкта під різними назвами. Це дозволить повноцінно реалізувати механізм наслідування для компонентів.

У разі необхідності компоненти можуть бути видалені з контексту додатку за допомогою методу:

```
static unregisterComponentInstance(component) {
  let prototype = component.constructor;
  while (prototype !== Component) {

this.#context.componentInstances.get(prototype.componentName).delete(component);
    prototype = Object.getPrototypeOf(prototype);
  }
}
```

Відбувається аналогічний циклічний прохід за прототипами батьківських класів.

Реєстрація екземплярів класів компонентів відбувається з метою їх подальшого отримання з контексту та використання:

```
static lookupComponentInstance(componentClass, container) {
  return this.lookupComponentInstances(componentClass, container)[0];
}

static lookupComponentInstances(componentClass, container) {
  const instances =
Array.from(this.#context.componentInstances.get(componentClass.componentName));
  if (container) {
    return instances.filter(instance => container.contains(instance.root));
  }
  return instances;
}
```

Для скорочення обсягів коду перший метод користується результатами наступного — більш узагальненого. У разі вказання класу матимемо можливість отримати об’єкти, що були раніше зареєстровані на поточній сторінці. Якщо передати параметр контейнеру, тоді додатково буде здійснюватися перевірка, чи знаходиться знайдений компонент усередині вказаного елемента.

Пропонуються також методи з інстанціювання — створення екземплярів компонентів потрібного класу, що базуються на вказаному кореневому вузлі, з їх подальшою автоматичною ініціалізацією:

```
static instantiateComponent(componentClass, root) {
  const component =
ObjectUtils.applyConstructor(componentClass.prototype.constructor, root);
  component.init();
  return component;
}

static instantiateComponents(componentClass, container, selector) {
  const roots = container.querySelectorAll(selector);
  const components = [];
  for (const root of roots) {
    components.push(this.instantiateComponent(componentClass, root));
  }
  return components;
}
```

Був використаний допоміжний клас із папки utils, що запустити динамічно визначений конструктор із передачею в нього параметрів. У нашому випадку це root — вузол у DOM-дереві, що слугуватиме основою компонента.

Уміст object-utils.js:

```
export default class ObjectUtils {
  static applyConstructor(ctor, args){
    return new (ctor.bind.apply(ctor, [null].concat(args)))();
  };
}
```

Основний клас Application дозволяє централізовано отримувати дані від сервера за допомогою виклику getServerData, що повертає об’єкт:

```
static getServerData() {
  return window.serverData;
}
```

Значення window.serverData встановлюється за допомогою генерації шаблонізатором на сервері тегу script усередині HTML-сторінки, що надається клієнту. Інших способів передачі значень із шаблонізатору до JS-файлів не існує.

Статичний метод isAuthenticated також спирається на значення, що повертає сервер:

```
static isAuthenticated() {
  return window.serverData['authenticated'] === true;
}
```

Далі в класі описані методи, що відповідають за виконання ініціалізації:

```
static init() {
  if (this.#initialized) {
    throw new Error('Application is already initialized.');
  }
  this.#loadComponents();
```

```
  this.#loadController();
  this.#initialized = true;
}
```

У першу чергу відбувається перевірка стану ініціалізованості клієнтської частини додатку шляхом використання приватної статичної змінної initialized. Це є необхідним для уникнення повторних викликів методу.

Якщо init виконується вперше — відбувається запуск методу loadComponents для завантаження екземплярів зареєстрованих раніше класів компонентів:

```
static #loadComponents() {
  const selector = Array.from(this.#context.components.keys())
    .map(name => '.' + name)
    .join(',');

  if (selector) {
    document.querySelectorAll(selector).forEach(element => {
      for (const className of element.classList) {
        if (this.#isComponentClassName(className) && selector.includes('.' +
className)) {
          const constructor =
this.#context.components.get(className).prototype.constructor;
          ObjectUtils.applyConstructor(constructor, element);
          break;
        }
      }
    });

    for (const componentClass of this.#context.componentInstances.keys()) {
      for (const instance of
this.#context.componentInstances.get(componentClass)) {
        if (!instance.isInitialized()) {
          instance.init();
        }
      }
    }
  }
}
```

Алгоритм виконання цього методу виглядає таким чином:

1. формується селектор із використанням назв зареєстрованих компонентів;
2. отримуються DOM-елементи шляхом застосування раніше побудованого селектора за допомогою document.querySelectorAll;
3. аналізується кожен елемент: перебираються його класи, вказані в атрибуті class, щоб визначити, до якого класу компонента належить елемент; у разі успіху — відбувається інстанціювання;
4. створені об’єкти компонентів ініціалізуються.

Важливо зазначити, що приналежність до компонента під певною назвою в нашому випадку відбувається на основі CSS-класів, іменованих за BEM-методологією (block-element-modifier):

```
static #isComponentClassName(className) {
  return !className.includes('__') && !className.includes('--');
}
```

Критерій визначення може бути будь-яким. Наприклад: data-атрибут.

Також варто звернути увагу на те, що ініціалізація інстанційованих компонентів відбувається окремим кроком. Таке рішення було прийнято для уникнення можливих проблем із залежностями, що могли бути спричинені недетермінованістю порядку реєстрації екземплярів. Таким чином можемо вільно в init методі компонента встановлювати зв’язок із будь-яким підкомпонентом.

Після цього кроку визначається контролер поточної сторінки за допомогою методу:

```
static #loadController() {
  const controllerName = document.documentElement.getAttribute('data-
controller');
  if (controllerName !== null) {
    const controllerClass = this.#context.controllers.get(controllerName);
    if (controllerClass !== undefined) {
      ObjectUtils
        .applyConstructor(controllerClass.prototype.constructor)
        .attach();
    } else {
      throw new Error(`Controller "${controllerName}" is undefined.`);
    }
  }
}
```

У ньому відбувається звернення до атрибута data-controller, що встановлюватиметься сервером для тегу html. Якщо назва контролера буде знайдена серед зареєстрованих — його об’єкт буде створений. На ньому одразу запуститься метод attach, що розпочне керування контролером цієї сторінки.

Якщо необхідний контролер не був зареєстрований — з’явиться помилка про його відсутність.

Також передбачена опція, коли сервер не визначає контролер для сторінки взагалі. У такому випадку пошук у контексті не відбувається.

Наступний за важливістю клас у розробленому фреймворку це Component, що описаний у component.js усередині папки components. Він опи-

сує абстрактний компонент, від якого відбуватиметься успадковуватимуться усі інші компоненти.

На початку опису класу визначені статичні змінні для зручності використання в коді:

```
static COMPONENT_PROPERTY = 'component';
static HANDLERS_SEPARATOR = ',';
```

Перша визначає назву поля, за яким буде зберігатися об’єкт компонента всередині DOM-елемента. Другий — встановлює роздільник, використовуючи який можна вказати декілька типів подій для одного обробника.

Конструктор класу має такий вигляд:

```
constructor(root) {
  this._root = root;
  this._root[this.constructor.COMPONENT_PROPERTY] = this;
  this.#handlers = new Map();
  this.#nestedComponents = new Set();
  Application.registerComponentInstance(this);
}
```

Він приймає вузол, на якому базуватиметься створюваний компонент. Він одразу прописується як кореневий. Далі створюється посилання на цей об’єкт шляхом встановлення значення поля під назвою, що вказана у відповідній статичній змінній. Після цього відбувається створення об’єктів — карти для handlers та множини для nestedComponents. Перший буде зберігати обробники подій, що реєструються. У свою чергу за допомогою другого підтримується відслідковування вкладених компонентів. Це дасть у подальшому можливість проводити коректне видалення.

Базова ініціалізація має вигляд:

```
init() {
  if (this._initialized) {
    throw new Error('Component is already initialized.');
  }
  this._initialized = true;
}
```

Її повинні будуть викликати всі нащадки за допомогою super.init(). В іншому випадку доведеться прописувати цю логіку вручну.

Варто зазначити, що в коді фреймворку можуть використовуватися префікси у вигляді нижнього підкреслення перед назвами методів та полів. Вони

вказують на те, що ці елементи можуть бути використані нащадками. Ззовні звертатися до них не рекомендується.

Наступним методом для розгляду є _defineHandlers:

```
_defineHandlers(element, ...handlers) {
  if (!element || Object.keys(handlers).length === 0) {
    return;
  }

  for (const handler of handlers) {
    for (const eventTypes in handler) {
      for (const eventType of
eventTypes.split(this.constructor.HANDLERS_SEPARATOR)) {
        element.addEventListener(eventType, handler[eventTypes]);
      }
    }
  }

  if (this.#handlers.has(element)) {
    this.#handlers.set(element, this.#handlers.get(element).concat(handlers));
  } else {
    this.#handlers.set(element, handlers);
  }
}
```

Його задача — виконувати реєстрацію обробників подій на елементи із збереженням інформації про це всередині об’єкта компонента, щоб в разі його видалення провести коректне очищення. Варто зазначити, що підтримується реєстрація обробників на будь-які елементи — навіть ті, що не знаходяться в межах компонента. Наприклад — об’єкт window. Досить часто на нього треба зареєструвати обробник події натискання кнопки миші, що вказувала б на бажання користувача закрити модальне вікно. За допомогою _defineHandlers — це проста задача.

Із особливостей роботи методу: можна зареєструвати одразу декілька обробників шляхом передачі об’єктів із ключами, які вказують на типи подій:

```
this._defineHandlers(window, {
  'click': event => {
    console.log('Example 1');
  }
}, {
  'keypress': event => {
    console.log('Example 2');
  }
});
```

Також можна перелічити всі необхідні події для обробки через кому у вигляді рядку:

```
this._defineHandlers(window, {
  'click,keypress': event => {
```

```
    console.log('Example');
  }
});
```

Для видалення зареєстрованих обробників можна скористатися методом:

```
_undefineHandlers(element, ...handlers) {
  for (const handler of handlers) {
    for (const eventTypes in handler) {
      for (const eventType of
eventTypes.split(this.constructor.HANDLERS_SEPARATOR)) {
        element.removeEventListener(eventType, handler[eventTypes]);
      }
    }
  }
  this.#handlers.delete(element)
}
```

Виклик буде впливати лише на ті, що були зареєстровані за допомогою _defineHandlers. Більш того варто передавати ті ж самі об’єкти з описаними всередині обробниками.

Для отримання посилання на елемент усередині компонента застосовуються методи:

```
#initElement(element, handlers) {
  this._defineHandlers(element, ...handlers);
  return element;
}

_initRoot(...handlers) {
  return this.#initElement(this._root, handlers);
}

_initElement(from, selector, ...handlers) {
  return this.#initElement(from.querySelector(selector), handlers);
}

_initElements(from, selector, ...handlers) {
  const elements = from.querySelectorAll(selector);
  for (const element of elements) {
    this.#initElement(element, handlers);
  }
  return elements;
}
```

Перший, що є приватним — для внутрішнього використання. Таким чином уникаємо повторень коду. Задача, що виконується — зареєструвати обробники, якщо такі вказані, та повернути посилання на елемент.

_initRoot — викликається в разі необхідності реєстрації обробників подій безпосередньо на кореневому вузлі компонента.

_initElement — виконає пошук елемента, відштовхуючись від значення змінної from за допомогою селектора selector. Обробники можуть бути вказані за необхідності.

Аналогічно — _initElements. Цей метод повертає NodeList, що містить шукані елементи. Зручно використовувати для реєстрації одного й того ж обробника для декількох DOM-вузлів.

Для отримання посилань на підкомпоненти застосовуються такі методи:

```
_initSubcomponent(componentClass, from, selector, ...handlers) {
  let subcomponent = this._initElement(from, selector,
...handlers)[this.constructor.COMPONENT_PROPERTY];
  if (!subcomponent) {
    subcomponent = Application.instantiateComponent(componentClass,
from.querySelector(selector));
  }

  if (!subcomponent.isInitialized()) {
    subcomponent.init();
  }
  this.#nestedComponents.add(subcomponent);

  return subcomponent;
}

_initSubcomponents(componentClass, from, selector, ...handlers) {
  let subcomponents = Array.from(this._initElements(from, selector,
...handlers)).map(element => element[this.constructor.COMPONENT_PROPERTY]);
  if (subcomponents.length === 0) {
    subcomponents = Application.instantiateComponents(componentClass, from,
selector);
  }

  for (const subcomponent of subcomponents) {
    if (!subcomponent.isInitialized()) {
      subcomponent.init();
    }
    this.#nestedComponents.add(subcomponent);
  }
  return subcomponents;
}
```

Вони мають аналогічний принцип виконання, але додатково треба вказати клас компонента, до якого намагаємося отримати доступ. Якщо за вказаними налаштуваннями не вдалося знайти компоненти — відбудеться спроба створити їх та ініціалізувати.

Корисним методом під час опису обробників є _callOuterHander:

```
_callOuterHandler(handler, ...args) {
  if (handler) {
    handler(...args);
  }
}
```

Він дозволяє виконати виклик функції зворотного виклику на випадок, якщо та була визначена. Підтримується передача параметрів.

Для коректного видалення підкомпонентів передбачений метод:

```
removeSubcomponent(subcomponent) {
  this.#nestedComponents.delete(subcomponent);
  subcomponent.remove();
}
```

Метод remove, що викликається має такий опис:

```
remove() {
  for (const element of this.#handlers.keys()) {
    this._undefineHandlers(element, ...this.#handlers.get(element));
  }

  for (const component of this.#nestedComponents) {
    this.removeSubcomponent(component);
  }

  Application.unregisterComponentInstance(this);
  this._root.remove();
}
```

Він узагальнює собою процес видалення компонента: знищуються раніше зареєстровані обробники подій, видаляються підкомпоненти, після чого — скасовується реєстрація компонента в додатку. І в кінці — видалення самого вузла.

Важливо зазначити, що видалення відбуватиметься рекурсивно — воно спочатку торкнеться підкомпонентів. Варто пам’ятати, що в разі існування підкомпонента всередині — на нього обов’язково треба отримати посилання, інакше — рекурсія не спрацює.

Цей випадок може бути вирішений за допомогою рекурсивного обходу елементів DOM, проте це є дуже неефективним, тому краще дбати про отримання всіх необхідних посилань, якщо компонент треба буде видаляти.

Властивості для отримання інформації:

```
  get nestedComponents() {
    return this.#nestedComponents;
  }

  static get componentName() {
    return 'component';
  }

  get componentName() {
    return this.constructor.componentName;
  }
```

```
  isInitialized() {
    return this._initialized;
  }
```

Найбільш важливим серед них є нестатичний метод componentName. Він визначає звернення до статичної змінної відповідного класу, щоб отримати назву компонента за екземпляром. Для нащадків обов'язковим для визначення є статичний метод componentName, що дозволить ідентифікувати об'єкти. Цей підхід доводиться застосовувати з огляду на використання мініфікації, яка не зберігає назви функцій та класів.

Аналогічно був визначений контролер абстрактної сторінки у файлі controller.js, що знаходиться в папці controllers:

```
import Application from '../application';

export default class Controller {
  attach() {
  }

  static controllerName() {
    return 'controller';
  }

  get controllerName() {
    return this.constructor.controllerName;
  }
}

Application.registerController(Controller);
```

За допомогою статичного методу controllerName визначається назва, що не буде втрачена після застосування мініфікатора. Відповідний нестатичний метод буде звертатися до цього значення. Нащадки класу Controller повинні визначати свої статичні методи, щоб їх можна було істанціювати.

Наявний порожній метод attach. Він буде перевизначатися нащадками.

Одразу після визначення контролера відбувається його реєстрація в додатку, тому він може бути вказаний у якості обробника сторінок.

Визначення точок входу не менш важливе. Приклад опису окремої з них:

```
import Application from '../application';

import '../controllers/home-controller';
import '../controllers/shops-controller';

Application.init();
```

Їх можна передбачити декілька, після чого зробити відповідні налаштування для збиральника.

Можливості розробленого фреймворку були використані для реалізації клієнтської частини веб-додатку.

У першу чергу треба було з’ясувати, які компоненти інтерфейсу будуть використовуватися на сторінках сайту. Їх перелік представлений у табл. 3.1.

Таблиця 3.1 — Компоненти інтерфейсу клієнтської частини веб-додатку

Компонент	**Опис**
Modal	Може обгортати інші компоненти для їх відображення в модальному вікні, що здатне закриватися шляхом натискання кнопки або за межами виводу контенту
Alert	Повідомлення для користувача різного типу, ініціатором появи яких виступає сервер — формує відповідний уміст serverData
CookiePolicy	Являє собою невелике вікно, що сповіщає користувача про використання технології cookie, із чим він має погодитися. Відображатиметься до тих пір, поки не отримає згоду
Header	Проміжна ланка взаємодії з іншими компонентами. Його задачею є надання можливості встановлення обробників подій натискання на кнопки авторизації, каталогу, корзини, обраного
SearchBox	Пошуковий рядок, що працює в асинхронному режимі, відображає товари — результати пошуку
Offer	Елемент, що дозволяє перемикатися між різними типами пропозицій товарів, представлених у вигляді окремих слайдерів
Card	Картка товару, реагує на натискання кнопки купівлі та обрання
Cart	Керує відображенням корзини, дозволяє додавати, змінювати кількість та видаляти з неї товар
CartItem	Окрема функціональна частина корзини, її підкомпонент, що може змінювати кількість продукції, за яку відповідає, видалятися за натисканням кнопки
GuestFavorites	Вікно, у якому розміщується обраний користувачем товар. Буде відображатися лише для неавторизованих. Для авторизованих передбачена окрема сторінка

Продовження таблиці 3.1

Компонент	**Опис**
GuestFavorite	Окрема функціональна одиниця вікна з обраними товарами. Може видалятися з переліку в результаті натискання користувачем відповідної кнопки
Catalog	Вікно, що повинно з’являтися в результаті натискання відповідної кнопки всередині Header
CatalogProduct	Функціональна сторінка товару, реагує на купівлю та додавання в обране
Shops	Керує фрагментом інформаційної сторінки, яка призначена для надання відомостей про відділи мережі та показує їх на картах Google
Counter	Лічильник, підтримує безпосередній ввід цілочисельного значення або його зміну на одиницю в сторону зменшення або збільшення
RangeSlider	Дозволяє визначати інтервал для змінної, що після завершення вводу користувачем потрапляє до адресного рядку, що спричиняє перехід
Paginator	Дозволяє змінювати розмір сторінки пагінації на довільний. Після внесення змін перехід до обраної конфігурації відбувається автоматично
Filter	Виконує фільтрацію компонентів, що відображаються, на основі вводу, що виконується за допомогою підкомпонентів
SortControl	Визначає напрямок сортування певної змінної
UpButton	Дозволяє користувачу швидко прогорнути сторінку вгору. З’являється в разі значного відхилення від верхівки сторінки
AdminHeader	Забезпечує керування інтерфейсом адміністратора за рахунок надання обробників на натискання елементів
AdminSidebar	Бокова панель інтерфейсу адміністратора. Виконує задачу із відображення елементів меню та підменю
AdminControls	Містить елементи керування таблицею із даними для адміністратора
AdminTable	Таблиця, що містить інформацію про окремий тип об’єктів, над якими адміністратор здійснює керування
OrderTable	Нащадок інформаційної таблиці, що адаптований під відображення відомостей про замовлення із можливістю внесення змін до його статусу

Компоненти, що являють собою форми, із якими буде взаємодіяти звичайний користувач, описані в табл. 3.2.

Таблиця 3.2 — Користувацькі форми

Назва	**Опис**
Form	Батьківський клас для форм, містить методи для виконання валідації вхідних даних
FormWithPassword	Форма, що містить поле для введення пароля. Додає обробку подій натискання на кнопку, що призводять до його приховання або відображення
LoginForm	Форма авторизації користувача
SignupForm	Форма реєстрації користувача
AuthForm	Форма авторизації, що містить у собі підкомпоненти у вигляді форм авторизації та реєстрації, між якими можна виконувати перемикання
ProfileForm	Форма, що дозволяє вносити зміни до користувацьких даних
CheckoutForm	Форма для створення замовлення
FeedbackForm	Форма надіслання відгуку користувача

Форми адміністратора описані в табл. 3.3.

Назва	**Опис**
ModelForm	Батьківський клас для форм, що призначені для створення або редагування об’єктів, якими керує адміністратор
CategoryForm	Форма для категорій товарів
TypeForm	Форма для типів товарів
ProductForm	Форма для товарів
VendorForm	Форма для постачальників
UserForm	Форма для користувачів
ShopForm	Форма для відділів мережі
SlidesForm	Форма для слайдів головної сторінки
SettingsForm	Форма налаштувань сайту

Програмний код реалізації основних компонентів наведений у додатку А. Основні контролери, що їх використовують, описані в додатку Б. У додатку передбачено дві точки входу: user та admin — відповідно до ролей (див. додаток В).

3.2 Реалізація серверної частини

Вона базується на використанні Laravel. Під час опису реалізації будемо звертати увагу на створювані сутності, передбачені в рамках цього фреймворку.

Для початку роботи над проектом була виконана команда:

```
composer create-project laravel/laravel fanzoo-web-store
```

Після того, як відбулося завантаження файлів фреймворку також треба запустити всередині папки проекту:

```
npm install
```

Це дозволить встановити залежності, що відповідають за генерацію клієнтської частини.

Далі треба виконати налаштування змінних оточення, що описані у файлі .env, а саме: вказати значення для APP_NAME та ввести інформацію, необхідну для підключення до бази даних.

За замовчуванням сторінки проекту доступні з префіксом /public, що є незручним. Оскільки плануємо використовувати веб-сервер Apache — створимо в директорії проекту файл без назви із розширенням htaccess такого вмісту:

```
RewriteEngine On
RewriteRule (.*) public/$1
```

Це дозволить виконувати веб-сервером внутрішні перенаправлення, що зроблять контент доступним за більш простим посиланням, що не містить префіксів.

Використовуючи phpMyAdmin виконаємо команду для створення бази даних, до якої будемо підключатися:

```
CREATE DATABASE fanzoo CHARACTER SET utf8 COLLATE utf8_general_ci;
```

Перейдемо до створення моделей та міграцій. Для цього за допомогою командного рядка виконуємо запити виду:

```
php artisan make:model Role -m
php artisan make:model Category -m
php artisan make:model Type -m
```

...

Ключ -m вказує на необхідність створення відповідного файлу міграції, у якому задається структура окремої таблиці.

Для категорії вона матиме вигляд:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('categories', function (Blueprint $table) {
            $table->unsignedTinyInteger('id')->autoIncrement();
            $table->string('name', 32)->unique();
            $table->text('description')->nullable();
            $table->string('photo')->nullable();
            $table->string('icon')->nullable();
            $table->timestamps();
            $table->softDeletes();
        });
    }

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

У методі up описується, таблиця під якою назвою створюється, якими колонками володітиме та які накладаються обмеження. У свою чергу down виконується під час

Виконавши опис усіх таблиць у файлах міграцій отримаємо структуру таблиць бази даних (див. рис. 3.2).

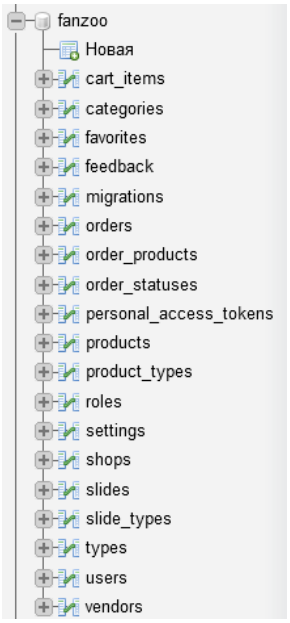

Рисунок 3.2 — Структура таблиць бази даних

Можемо бачити, що в ній представлені додаткові таблиці: migrations та personal_access_tokens. Вони необхідні для внутрішнього використання фреймворком.

Для заповнення бази даних тестовою інформацією необхідно багато часу, тому автори Laravel знайшли спосіб полегшити цей процес. Вирішенням стало використання Seeder та Factory-класів, що займаються генерацією даних. Останні відповідають за опис значень, що будуть встановлюватися за замовчуванням. Разом із бібліотекою Faker заповнення бази даних стає просто задачею. Шаблон коду цих класів може бути згенерований шляхом виконання відповідних команд для artisan.

Ролі користувачів у нашому додатку є постійними, тому їх Seeder не матиме елементів рандомізації. Утім, можна застосувати для генерації акаунтів користувачів. Клас UserSeeder матиме вигляд:

```
class UserSeeder extends Seeder
{
    public function run()
    {
        if (env('APP_DEBUG')) {
            $adminRoleID = Role::where('name', 'admin')->value('id');
            $userRoleID = Role::where('name', 'user')->value('id');

            User::factory(3)->create(['role_id' => $adminRoleID]);
            User::factory(97)->create(['role_id' => $userRoleID]);
            User::factory(25)->deleted()->create(['role_id' => $userRoleID]);
```

```
            User::factory(5)->withoutPhoneNumber()->create(['role_id' =>
$userRoleID]);
        } else {
            foreach (DatabaseSeeder::$default['users'] as $user) {
                if (isset($user['role'])) {
                    $user['role_id'] = Role::where('name', $user['role'])-
>value('id');
                }
                if (isset($user['password'])) {
                    $user['password'] = bcrypt($user['password']);
                }

                User::factory()->create(wrap_model_data($user, [
                    'id',
                    'role_id',
                    'name',
                    'email',
                    'password',
                    'phone_number',
                    'remember_token',
                    'created_at',
                    'updated_at',
                    'deleted_at',
                ]));
            }
        }
    }
}
```

Фабрика зі створення фейкових користувачів:

```
class UserFactory extends Factory
{
    public function definition()
    {
        $firstName = $this->faker->firstName();
        $lastName = $this->faker->lastName();
        $uniqueSuffix = $this->faker->unique()->word();
        $domain = $this->faker->safeEmailDomain();
        return [
            'name' => "{$firstName} {$lastName}",
            'email' => Str::lower($firstName) . '.' . Str::lower($lastName) .
".{$uniqueSuffix}@{$domain}",
            'password' =>
'$2y$10$k5RgeL5DGVuFV94ReetVi.QM8RjB6svgytPE3/5FzeSwmXo9ThuBK', //
bcrypt('password'),
            'phone_number' => $this->faker->regexify('\+\d{3} \d{2} \d{3} \d{2}
\d{2}'),
            'remember_token' => Str::random(10),
        ];
    }

    public function deleted()
    {
        return $this->state(function (array $attributes) {
            return [
                'deleted_at' => now(),
            ];
        });
    }

    public function withoutPhoneNumber()
    {
```

```
        return $this->state(function (array $attributes) {
            return [
                'phone_number' => null,
            ];
        });
    }
}
```

У разі, якщо змінна оточення APP_DEBUG буде рівною false — використання даних буде відбуватися з json-файлу, що заздалегідь буде містити структурований опис конфігурації цілої мережі. Його завантаження відбуватиметься в DatabaseSeeder'і.

3.3 Тестування веб-додатку

Було перевірено програмну реалізацію на відповідність функціональним вимогам, що були окреслені під час формулювання постановки задачі.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра був здійснений огляд літературних джерел за тематикою розробки веб-додатків, призначених для підтримки бізнес-діяльності мереж магазинів. Опрацьована інформація допомогла сформувати всебічне уявлення про основні аспекти створення Інтернет-магазину із застосуванням сучасних технологій та інструментів.

Були успішно проаналізовані актуальні на сьогоднішній день підходи до розробки веб-додатків, сформовані під впливом тенденцій розвитку галузі.

Під час проведення інформаційного огляду був здійснений аналіз уже існуючих аналогічних Інтернет-магазинів зі схожою тематикою на предмет виявлення затребуваних користувачами основних складових та підтримуваних функцій, що в подальшому дозволило сформувати постановку задачі у вигляді деталізованого списку вимог до реалізації.

Базуючись на знаннях та особистому досвіді, що були отримані під час проходження навчання, був обраний перелік інструментів та методів, що дозволили вирішити поставлену задачу найкращим чином, а сформований невеликий JavaScript-фреймворк, в основі якого був закладений компонентний підхід, показав свою ефективність.

Таким чином мета, визначена на початку дипломної роботи, була успішно досягнута: був реалізований веб-додаток для підтримки діяльності мережі зоомагазинів «FanZoo», що володіє основними складовими для ведення бізнесу в мережі Інтернет. Серед них: реалізований контекстний пошук, сортування і фільтрація каталогу товарів. Також наявна можливість використання корзини, формування переліку обраних товарів і оформлення замовлення. Адміністратор може виконувати керування контентом та обробляти замовлення.

За результатами тестування сайт успішно функціонує без помилок та однаково відображається в найбільш поширених веб-браузерах, таких, як Google Chrome, Mozilla Firefox, Microsoft Edge із версіями, що датуються 2018 роком та новішими, відповідно до раніше сформованих вимог до реалізації.

СПИСОК ЛІТЕРАТУРИ

1. Smith A. U. B. Resale Product and Accessory Opportunities for Audiology Practices // Seminars in Hearing. — Thieme Medical Publishers, 2019. — Т. 40. — №. 03. — С. 232-244.

2. Nagyová A. User experience design of a book website / A. Nagyová. — 2019.

3. Deshmukh S. Building a single page application web front-end for e-learning site / S. Deshmukh, D. Mane, A. Retawade. — 2019.

4. Berggren L. The User Experience of the Installation Process of Progressive Web Applications: A User Test / L. Berggren // USCCS 2022. — С. 1.

5. Kaur A. What is Single Page Application (SPA)? Pros and Cons with Examples [Електронний ресурс] / Arshpreet Kaur. — 2021. — Режим доступу до ресурсу: https://www.netsolutions.com/insights/single-page-application/.

6. Patel T. A Review on Content Management Systems of Web Development / T. Patel, S. Mittal, N. Kumar Awadhiya // Int. J. Comput. Sci. Trends Technol. — 2019. — Вип. 7, № 2.

7. Haris N. A. PHP frameworks usability in web application development / N. A. Haris, N. Hasim // Int. J. Recent Technol. Eng. — 2019. — Вип. 8, № 3 Special Issue.

8. O'Kane M. A Web-Based Introduction to Programming: Essential Algorithms, Syntax, and Control Structures Using PHP, HTML, and MariaDB/MySQL / M. O'Kane. — 2019.

9. Ahmed M. R. A literature review on NoSQL database for big data processing / M. R. Ahmed, M. A. Khatun, M. A. Ali, K. Sundaraj // Int. J. Eng. Technol. — 2018. — Вип. 7, № 2.

10. Most Popular Backend Frameworks — 2012/2021 — New Update [Електронний ресурс] — 2021. — Режим доступу до ресурсу: https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2021/.

11. Stauffer M. Laravel: Up and Running A Framework for Building Modern PHP Apps / M. Stauffer. — 2019.

12. Dockins K. Design Patterns in PHP and Laravel / K. Dockins. — 2017.

13. Libby A. Introducing Dart Sass / A. Libby. — 2019.

14. ReactDOM [Електронний ресурс] — Режим доступу до ресурсу: https://en.reactjs.org/docs/react-dom.html.

15. Web Components [Електронний ресурс] — Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/Web_Components.

16. Сивоконь В. В. Організація розробки клієнтської частини багатосторінкового додатку мовою JavaScript / В. В. Сивоконь, О. Б. Проценко // Інформатика, математика, автоматика IMA::2022: матеріали та програма міжнародної наукової конференції молодих учених, м. Суми, 18-22 квітня 2022 р. — С. 37-38.

ДОДАТКИ

Додаток А. Лістинги основних компонентів клієнтської частини

Cart

```
import Application from '../application';
import Component from './component';
import DOMUtils from '../utils/dom-utils';
import CartItem from './cart-item';

export default class Cart extends Component {
  #cartItems;
  #continueButton;
  #orderButton;

  removeItemHandler;
  updateHandler;
  continueButtonClickHandler;

  static get componentName() {
    return 'cart';
  }

  init() {
    super.init();
    this.#continueButton = this._initElement(this._root, '.cart__continue-
button', {
      'click': event => {
        this._callOuterHandler(this.continueButtonClickHandler, event);
      }
    });
    this.#orderButton = this._initElement(this._root, '.cart__order-button');
    this.#initSubcomponents();
  }

  #initSubcomponents() {
    this.#cartItems = this._initSubcomponents(CartItem, this._root, '.cart-
item');
    for (const cartItem of this.#cartItems) {
      cartItem.deleteButtonClickHandler = (event, productID) => {
        this.removeItem(productID);
      }
      cartItem.changeQuantityHandler = (productID, newValue, oldValue) => {
        this.setItemQuantity(productID, newValue);
      }
    }
    DOMUtils.setClass(this.#orderButton, this.quantity === 0, 'cart__order-
button--hidden');
  }

  #update(endpoint, parameters, callback, ...args) {
    for (const cartItem of this.#cartItems) {
      cartItem.disable(true);
    }

    const csrfToken = DOMUtils.getCSRFToken();
    const request = new XMLHttpRequest();
    request.open(
      'POST',
      `${window.location.protocol}//${window.location.hostname}${endpoint}`
    );
```

```
    request.setRequestHeader('X-CSRF-TOKEN', csrfToken);
    request.setRequestHeader('Accept', 'text/html');
    request.setRequestHeader('Content-Type', 'application/json');

    request.onload = () => {
      for (const cartItem of this.#cartItems) {
        this.removeSubcomponent(cartItem);
      }

      const parser = new DOMParser();
      const doc = parser.parseFromString(request.response, 'text/html');
      this._root.querySelector('.cart__content').outerHTML =
doc.querySelector('.cart__content').outerHTML;

      Application.instantiateComponents(CartItem, this._root, '.cart-item');
      this.#initSubcomponents();

      this._callOuterHandler(callback, ...args);
      this._callOuterHandler(this.updateHandler, this.quantity);

      if (this.quantity === 0) {
        if (window.location.pathname === '/checkout') {

window.location.replace(`${window.location.protocol}//${window.location.hostname
}`);
        }
      }
    }

    request.send(JSON.stringify(parameters));
  }

  addItem(productID) {
    this.#update('/cart/add-item', {
      'productID': productID,
    });
  }

  removeItem(productID) {
    this.#update('/cart/remove-item', {
      'productID': productID,
    }, this.removeItemHandler, productID);
  }

  setItemQuantity(productID, quantity) {
    if (quantity <= 0) {
      this.removeItem(productID);
    } else {
      this.#update('/cart/set-item-quantity', {
        'productID': productID,
        'quantity': quantity,
      });
    }
  }

  hasItem(productID) {
    for (const cartItem of this.#cartItems) {
      if (cartItem.productID === productID) {
        return true;
      }
    }
    return false;
  }
```

```
  get quantity() {
    let quantity = 0;
    for (const cartItem of this.#cartItems) {
      quantity += cartItem.quantity;
    }
    return quantity;
  }
}

Application.registerComponent(Cart);
```

CartItem

```
import Application from '../application';
import Component from './component';
import Counter from './counter';

export default class CartItem extends Component {
  #productID;
  #quantityCounter;
  #deleteButton;

  deleteButtonClickHandler;
  changeQuantityHandler;

  static get componentName() {
    return 'cart-item';
  }

  init() {
    super.init();

    this.#productID = +this._root.getAttribute('data-id');
    this.#deleteButton = this._initElement(this._root, '.cart-item__delete-
button', {
      'click': event => {
        this._callOuterHandler(this.deleteButtonClickHandler, event,
this.#productID);
      }
    });
    this.#quantityCounter = this._initSubcomponent(Counter, this._root, '.cart-
item__quantity');
    this.#quantityCounter.changeHandler = (newValue, oldValue) => {
      this._callOuterHandler(this.changeQuantityHandler, this.#productID,
newValue, oldValue);
    }
  }

  disable(disable) {
    this.#deleteButton.disabled = disable;
    this.#quantityCounter.disable(disable);
  }

  get productID() {
    return this.#productID;
  }

  get quantity() {
    return this.#quantityCounter.value;
  }
}
```

```
Application.registerComponent(CartItem);
```

GuestFavorites

```
import Application from '../application';
import Component from './component';
import DOMUtils from '../utils/dom-utils';
import GuestFavorite from './guest-favorite';

export default class GuestFavorites extends Component {
  #items;
  #continueButton;
  #saveButton;

  continueButtonClickHandler;
  saveButtonClickHandler;
  removeItemHandler;
  updateHandler;

  static get componentName() {
    return 'guest-favorites';
  }

  init() {
    super.init();

    this.#continueButton = this._initElement(this._root, '.guest-
favorites__continue-button', {
      'click': event => {
        event.stopImmediatePropagation();
        this._callOuterHandler(this.continueButtonClickHandler, event);
      }
    });
    this.#saveButton = this._initElement(this._root, '.guest-favorites__save-
button', {
      'click': event => {
        event.stopImmediatePropagation();
        this._callOuterHandler(this.saveButtonClickHandler, event);
      }
    });
    this.#initSubcomponents();
  }

  #initSubcomponents() {
    this.#items = this._initSubcomponents(GuestFavorite, this._root, '.guest-
favorite');
    for (const item of this.#items) {
      item.deleteButtonClickHandler = (event, productID) => {
        this.removeItem(productID);
      }
    }
    DOMUtils.setClass(this.#saveButton, this.quantity === 0, 'guest-
favorites__save-button--hidden');
  }

  #update(endpoint, parameters, callback, ...args) {
    const csrfToken = DOMUtils.getCSRFToken();
    const request = new XMLHttpRequest();
    request.open(
      'POST',
      `${window.location.protocol}//${window.location.hostname}${endpoint}`
    );
    request.setRequestHeader('X-CSRF-TOKEN', csrfToken);
```

```
    request.setRequestHeader('Accept', 'text/html');
    request.setRequestHeader('Content-Type', 'application/json');

    request.onload = () => {
      for (const item of this.#items) {
        this.removeSubcomponent(item);
      }

      const parser = new DOMParser();
      const doc = parser.parseFromString(request.response, 'text/html');
      this._root.querySelector('.guest-favorites__content').outerHTML =
doc.querySelector('.guest-favorites__content').outerHTML;

      Application.instantiateComponents(GuestFavorite, this._root, '.guest-
favorite');
      this.#initSubcomponents();

      this._callOuterHandler(callback, ...args);
      this._callOuterHandler(this.updateHandler, this.quantity);
    }

    request.send(JSON.stringify(parameters));
  }

  addItem(productID) {
    this.#update('/favorites/add-item', {
      'productID': productID,
    });
  }

  removeItem(productID) {
    this.#update('/favorites/remove-item', {
      'productID': productID,
    }, this.removeItemHandler, productID);
  }

  hasItem(productID) {
    for (const item of this.#items) {
      if (item.productID === productID) {
        return true;
      }
    }
    return false;
  }

  get quantity() {
    return this.#items.length;
  }
}

Application.registerComponent(GuestFavorites);
```

GuestFavorite

```
import Application from '../application';
import Component from './component';

export default class GuestFavorite extends Component {
  #productID;
  #deleteButton;

  deleteButtonClickHandler;
```

```
  static get componentName() {
    return 'guest-favorite';
  }

  init() {
    super.init();

    this.#productID = +this._root.getAttribute('data-id');
    this.#deleteButton = this._initElement(this._root, '.guest-favorite__delete-
button', {
      'click': event => {
        this._callOuterHandler(this.deleteButtonClickHandler, event,
this.#productID);
      }
    });
  }

  get productID() {
    return this.#productID;
  }
}

Application.registerComponent(GuestFavorite);
```

Form

```
import Application from '../../application';
import Component from '../component';
import DOMUtils from '../../utils/dom-utils';
import HandlerUtils from '../../utils/handler-utils';

export default class Form extends Component {
  static UNIQUE_PROPERTY = 'unique-error';
  #controls = new Map();

  _defineControl(type, name, accessor, initialValue, validator,
uniquenessValidator) {
    let control = this._initElement(this._root, `[name="${name}"]`);
    let handlers = {};
    let validators = [];

    switch (type) {
      default:
      case 'email':
      case 'password':
      case 'text':
      case 'url':
      case 'tel':
      case 'search':
      case 'textarea':
        if (validator) {
          const validationHandler =
            () => this._validate(
                control,
                accessor,
                validator['predicate'],
                validator['messageIfInvalid']
              );
          validators.push(validationHandler);

          if (uniquenessValidator) {
            const editingValue = accessor(control);
            const uniquenessHandler = HandlerUtils.debounce(
```

```
        () => {
          this._validateUnique(
            control,
            accessor,
            uniquenessValidator['type'],
            uniquenessValidator['endpoint'],
            uniquenessValidator['messageIfNonUnique'],
            uniquenessValidator['editing'],
            editingValue
          );
        },
        uniquenessValidator['delay']
      );

      handlers['blur'] = () => {
        if ((this._isValid(control) && accessor(control) === initialValue)
          || this._isUnique(control)) {
          validationHandler();
        }
      };

      validators.push(uniquenessHandler);
      handlers['input'] = () => {
        if (validationHandler()) {
          uniquenessHandler();
        }
      };
    } else {
      handlers['blur,input'] = validationHandler;
    }
  }
  break;
case 'number':
  break;
case 'datetime-local':
case 'date':
case 'week':
case 'month':
  break;
case 'radio':
case 'checkbox':
  break;
case 'color':
  break;
case 'file':
  if (validator) {
    validators.push(() => this._validate(
      control,
      accessor,
      validator['predicate'],
      validator['messageIfInvalid']
    ));

    handlers['change'] = () => {
      validators[0]();
    };
  }
  break;
case 'button':
case 'submit':
case 'image':
case 'reset':
  break;
```

```
      case 'select':
        break;
    }
    this._defineHandlers(control, handlers);

    if (initialValue === null || initialValue === undefined) {
      initialValue = accessor(control);
    }

    this.#controls.set(name, {
      'control': control,
      'accessor': accessor,
      'initialValue': initialValue,
      'validators': validators,
    });
    return control;
  }

  _getRow(element) {
    let row = element.parentNode;
    while (!DOMUtils.hasClass(row, 'form__row') && row !== document) {
      row = row.parentNode;
    }

    if (row === document) {
      return null;
    }
    return row;
  }

  _addError(row, message, unique) {
    const errorMessage = document.createElement('span');
    errorMessage.className = 'form__error';
    errorMessage.textContent = message;
    if (unique) {
      row[this.constructor.UNIQUE_PROPERTY] = 'error';
    }
    row.appendChild(errorMessage);
    DOMUtils.addClass(row, 'form__row--with-error');
  }

  _removeErrors(row) {
    row.querySelectorAll('.form__error')
      .forEach(element => element.remove());
    DOMUtils.removeClass(row, 'form__row--with-error');
    delete row[this.constructor.UNIQUE_PROPERTY];
  }

  _isValid(element) {
    return !DOMUtils.hasClass(this._getRow(element), 'form__row--with-error');
  }

  _isUnique(element) {
    return this._getRow(element)[this.constructor.UNIQUE_PROPERTY] ===
undefined;
  }

  _validate(element, accessor, validityPredicate, messageIfInvalid) {
    let row = this._getRow(element);

    if (row === null) {
      throw new Error('Validated element isn\'t inside form row.');
    }
```

```
    if (validityPredicate(accessor(element))) {
      this._removeErrors(row);
      return true;
    }

    // validity > uniqueness
    if (!this._isUnique(element)) {
      this._removeErrors(row);
    }

    if (!DOMUtils.hasClass(row, 'form__row--with-error')) {
      this._addError(row, messageIfInvalid, false);
    }
    return false;
  }

  _validateUnique(element, accessor, type, endpoint, errorMessage, editing,
editingValue) {
    if (!this._isValid(element)) {
      return;
    }
    const validContent = accessor(element);
    if (editing && validContent === editingValue) {
      return;
    }

    const csrfToken = document.querySelector('meta[name="csrf-token"]').content;
    const request = new XMLHttpRequest();
    request.open(
      'POST',
      `${window.location.protocol}//${window.location.hostname}${endpoint}`
    );
    request.setRequestHeader('X-CSRF-TOKEN', csrfToken);
    request.setRequestHeader('Accept', 'application/json');
    request.setRequestHeader('Content-Type', 'application/json');
    request.onload = () => {
      if (accessor(element) === validContent) {
        const response = JSON.parse(request.response);
        const row = this._getRow(element);

        this._removeErrors(row);

        if (response['exists']) {
          this._addError(row, errorMessage, true);
        } else {
          this._removeErrors(row);
          delete row[this.constructor.UNIQUE_PROPERTY];
        }
      }
    }
    request.send(JSON.stringify({
      [type]: validContent,
    }));
  }

  init() {
    super.init();

    for (const control of this.#controls.values()) {
      if (control['accessor'](control['control']) !== control['initialValue']) {
        for (const validator of control['validators']) {
          validator();
```

```
          if (!this._isValid(control['control'])) {
            break;
          }
        }
      }
    }

    this._root.querySelectorAll('.form__row')
      .forEach(row => {
        if (row.querySelector('.form__error')) {
          DOMUtils.addClass(row, 'form__row--with-error');
        }
      });

    this._initRoot({
      'submit': event => {
        if (DOMUtils.exists(event.currentTarget, '.form__row--with-error')) {
          event.preventDefault();
        } else {
          for (const control of this.#controls.values()) {
            for (const validator of control['validators']) {
              validator();
            }
          }
          if (DOMUtils.exists(event.currentTarget, '.form__row--with-error')) {
            event.preventDefault();
          }
        }
      }
    });
  }
}

Application.registerComponent(Form);
```

FormWithPassword

```
import DOMUtils from '../../utils/dom-utils';
import Application from '../../application';
import Form from './form';

export default class FormWithPassword extends Form {
  _passwordControl;
  _passwordControlButton;

  init() {
    super.init();

    if (!Application.isAuthenticated()) {
      this._passwordControl = this._initElement(this._root,
'[type="password"]');
      this._passwordControlButton = this._initElement(this._root,
'.form__password-control-button', {
        'click': event => {
          if (this._passwordControl.getAttribute('type') === 'password') {
            this._passwordControl.setAttribute('type', 'text');
            DOMUtils.removeClass(this._passwordControlButton, 'form__password-
control-button--hidden-password');
          } else {
            this._passwordControl.setAttribute('type', 'password');
            DOMUtils.addClass(this._passwordControlButton, 'form__password-
control-button--hidden-password');
          }
```

```
        }
      });
    }
  }
}

Application.registerComponent(FormWithPassword);
```

AuthForm

```
import Component from '../component';
import Application from '../../application';
import DOMUtils from '../../utils/dom-utils';
import LoginForm from './login-form';
import SignupForm from './signup-form';

export default class AuthForm extends Component {
  #loginForm;
  #signupForm;

  #loginFormTab;
  #signupFormTab;

  static get componentName() {
    return 'auth-form';
  }

  init() {
    super.init();

    this.#loginForm = Application.lookupComponentInstance(LoginForm,
this._root);
    this.#signupForm = Application.lookupComponentInstance(SignupForm,
this._root);

    this.#loginFormTab = this.#loginForm.root.parentNode;
    this.#signupFormTab = this.#signupForm.root.parentNode;

    this.#loginForm.toSignupFormButtonHandler = event => this.switchTo('signup-
form');
    this.#signupForm.toLoginFormButtonHandler = event => this.switchTo('login-
form');
  }

  switchTo(form) {
    switch (form) {
      default:
      case 'login-form':
        DOMUtils.addClass(this.#loginFormTab, 'auth-form__tab--active');
        DOMUtils.removeClass(this.#signupFormTab, 'auth-form__tab--active')
        break;
      case 'signup-form':
        DOMUtils.removeClass(this.#loginFormTab, 'auth-form__tab--active');
        DOMUtils.addClass(this.#signupFormTab, 'auth-form__tab--active');
        break;
    }
  }
}

Application.registerComponent(AuthForm);
```

LoginForm

```
import Application from '../../application';
```

```
import FormWithPassword from './form-with-password';
import ValidationUtils from '../../utils/validation-utils';

export default class LoginForm extends FormWithPassword {
  #toSignupFormButton;
  #toSignupFormButtonHandler;

  static get componentName() {
    return 'login-form';
  }

  constructor(root) {
    super(root);

    this._defineControl('email', 'email', e => e.value, '', {
      'predicate': value => ValidationUtils.EMAIL_REGEX.test(value),
      'messageIfInvalid': 'Некоректна адреса електронної пошти.',
    });
    this._defineControl('password', 'password', e => e.value, '', {
      'predicate': value => value.length > 0,
      'messageIfInvalid': 'Пароль не може бути порожнім.',
    });
  }

  init() {
    super.init();
    this.#toSignupFormButton = this._initElement(this._root, '.button--
secondary', {
      'click': event => {
        this._callOuterHandler(this.#toSignupFormButtonHandler, event);
      }
    });
  }

  set toSignupFormButtonHandler(handler) {
    this.#toSignupFormButtonHandler = handler;
  }
}

Application.registerComponent(LoginForm);
```

SignupForm

```
import Application from '../../application';
import FormWithPassword from './form-with-password';
import ValidationUtils from '../../utils/validation-utils';

export default class SignupForm extends FormWithPassword {
  #toLoginFormButton;
  #toLoginFormButtonHandler;

  static get componentName() {
    return 'signup-form';
  }

  constructor(root) {
    super(root);

    this._defineControl('text', 'name', e => e.value, '', {
      'predicate': value => value.length > 0,
      'messageIfInvalid': 'Це поле треба заповнити.',
    });
    this._defineControl('email', 'email', e => e.value, '', {
```

```
      'predicate': value => ValidationUtils.EMAIL_REGEX.test(value),
      'messageIfInvalid': 'Некоректна адреса електронної пошти.',
    }, {
      'type': 'email',
      'endpoint': '/account/check-email-existence',
      'messageIfNonUnique': 'Електронна пошта вже використовується.',
      'delay': 1000,
    });
    this._defineControl('password', 'password', e => e.value, '', {
      'predicate': value => value.length >= 8 && value.length <= 72 &&
value.toLowerCase() !== value && /\d+/.test(value),
      'messageIfInvalid': 'Пароль не відповідає вимогам.',
    });
  }

  init() {
    super.init();
    this.#toLoginFormButton = this._initElement(this._root, '.button--
secondary', {
      'click': event => {
        this._callOuterHandler(this.#toLoginFormButtonHandler, event);
      }
    });
  }

  set toLoginFormButtonHandler(handler) {
    this.#toLoginFormButtonHandler = handler;
  }
}

Application.registerComponent(SignupForm);
```

AdminControls

```
import Application from '../application';
import Component from './component';
import DOMUtils from '../utils/dom-utils';
import Paginator from './paginator';

export default class AdminControls extends Component {
  #ignoreElements = [];

  #filterButton;
  #filterMenu;
  #settingsButton;
  #settingsMenu;
  #columnCheckboxes;
  #paginator;

  columnCheckboxesChangeHandler;

  static get componentName() {
    return 'admin-controls';
  }

  init() {
    super.init();

    this.#filterButton = this._initElement(this._root, '.admin-controls__button-
-filter', {
      'click': event => {
        this.#showMenu(this.#filterMenu,
!this.#isMenuVisible(this.#filterMenu));
```

```
      }
    });
    this.#settingsButton = this._initElement(this._root, '.admin-
controls__button--settings', {
      'click': event => {
        this.#showMenu(this.#settingsMenu,
!this.#isMenuVisible(this.#settingsMenu));
      }
    });

    if (this.#filterButton) {
      this.#filterMenu = this._initElement(this._root, '.admin-controls__menu--
filter');
      this.#ignoreElements['filterMenu'] = [this.#filterButton];
      this._defineHandlers(window, {
        'click': event => {
          if (!this.#ignoreElements['filterMenu'].includes(event.target) &&
!this.#filterMenu.contains(event.target) &&
this.#isMenuVisible(this.#filterMenu)) {
            this.#showMenu(this.#filterMenu, false);
          }
        }
      });
    }
    if (this.#settingsButton) {
      this.#settingsMenu = this._initElement(this._root, '.admin-controls__menu-
-settings');
      this.#columnCheckboxes = this._initElements(this._root, '.admin-
controls__item-checkbox', {
        'click': event => {
          const id = event.currentTarget.id;
          this._callOuterHandler(this.columnCheckboxesChangeHandler,
+id.substring(id.indexOf('[') + 1, id.lastIndexOf(']')),
event.currentTarget.checked)
        }
      });
      this.#ignoreElements['settingsMenu'] = [this.#settingsButton];
      this._defineHandlers(window, {
        'click': event => {
          if (!this.#ignoreElements['settingsMenu'].includes(event.target) &&
!this.#settingsMenu.contains(event.target) &&
this.#isMenuVisible(this.#settingsMenu)) {
            this.#showMenu(this.#settingsMenu, false);
          }
        }
      });
    }
    this.#paginator = this._initSubcomponent(Paginator, this._root, '.admin-
controls__paginator');
  }

  #isMenuVisible(menu) {
    return DOMUtils.hasClass(menu, 'admin-controls__menu--visible');
  }

  #showMenu(menu, show) {
    DOMUtils.setClass(menu, show, 'admin-controls__menu--visible')
  }

  get statesOfColumnCheckboxes() {
    return Array.from(this.#columnCheckboxes).map(e => e.checked);
  }
}
```

```
Application.registerComponent(AdminControls);
```

AdminHeader

```
import Application from '../application';
import Component from './component';

export default class AdminHeader extends Component {
  #burgerButton;
  burgerButtonClickHandler;

  static get componentName() {
    return 'admin-header';
  }

  init() {
    super.init();

    this.#burgerButton = this._initElement(this._root, '.admin-header__burger',
{
      'click': event => {
        this._callOuterHandler(this.burgerButtonClickHandler, event);
      }
    });
  }
}

Application.registerComponent(AdminHeader);
```

AdminSidebar

```
import Application from '../application';
import Component from './component';
import DOMUtils from '../utils/dom-utils';

export default class AdminSidebar extends Component {
  #catalogSubmenu;
  #catalogSubmenuButton;

  static get componentName() {
    return 'admin-sidebar';
  }

  init() {
    super.init();

    this.#catalogSubmenu = this._initElement(this._root, '.admin-
sidebar__submenu');
    this.#catalogSubmenuButton = this._initElement(this._root, '.admin-
sidebar__item--with-submenu', {
      'click': event => {
        const newVisibilityState = !this.isSubmenuVisible();
        DOMUtils.setClass(this.#catalogSubmenu, newVisibilityState, 'admin-
sidebar__submenu--visible');
        DOMUtils.setClass(this.#catalogSubmenuButton, newVisibilityState,
'admin-sidebar__item--open');
      }
    });
  }

  show(show) {
    DOMUtils.setClass(this._root, show, 'admin-sidebar--visible');
  }
```

```
  isVisible() {
    return DOMUtils.hasClass(this._root, 'admin-sidebar--visible');
  }

  isSubmenuVisible() {
    return DOMUtils.hasClass(this.#catalogSubmenu, 'admin-sidebar__submenu--
visible');
  }
}

Application.registerComponent(AdminSidebar);
```

AdminTable

```
import Application from '../application';
import Component from './component';
import DOMUtils from '../utils/dom-utils';

export default class AdminTable extends Component {
  #columnHeadings;

  static get componentName() {
    return 'admin-table';
  }

  init() {
    super.init();

    this.#columnHeadings = this._initElements(this._root, '.admin-
table__heading');
  }

  showColumnByNumber(number, show) {
    if (number > this.#columnHeadings.length || number <= 0) {
      return;
    }

    const elements = this._root.querySelectorAll(`.admin-table__row > .admin-
table__cell:nth-child(${number})`);
    for (const element of elements) {
      DOMUtils.setClass(element, !show, 'admin-table__cell--hidden');
    }

    DOMUtils.setClass(this.#columnHeadings[number - 1], !show, 'admin-
table__heading--hidden');
  }
}

Application.registerComponent(AdminTable);
```

Modal

```
import Component from './component';
import Application from '../application';
import DOMUtils from '../utils/dom-utils';

export default class Modal extends Component {
  #ignoreElements = [];

  #contentHolder;
  #closeButton;
  #wrappedComponent;
  #wrappedComponentPreviousParentElement;
```

```
  closeButtonClickHandler;
  closeHandler;
  escapeHandler;

  static get componentName() {
    return 'modal';
  }

  init() {
    super.init();

    this.#contentHolder = this._initElement(this._root, '.modal__inner');
    this.#closeButton = this._initElement(this._root, '.modal__close-button', {
      'click': event => this.closeButtonClickHandler(event),
    });

    this._initRoot({
      'click': event => {
        if (this.isVisible() && !this.#contentHolder.contains(event.target) &&
!this.#ignoreElements.includes(event.target)) {
          this.closeHandler(event);
        }
      }
    });

    this._defineHandlers(window, {
      'keydown': event => {
        if (event.key === 'Escape') {
          this.escapeHandler(event);
        }
      }
    });
  }

  wrapComponent(component) {
    if (this.#wrappedComponent === component) {
      return;
    }
    this.unwrapComponent();

    this.#wrappedComponent = component;
    this.#wrappedComponentPreviousParentElement =
this.#wrappedComponent._root.parentNode;

    this.#contentHolder.appendChild(this.#wrappedComponent._root);
  }

  unwrapComponent() {
    if (this.#wrappedComponent !== undefined) {

this.#wrappedComponentPreviousParentElement.appendChild(this.#wrappedComponent._
root);
    }
    this.#wrappedComponent = undefined;
    this.#wrappedComponentPreviousParentElement = undefined;
  }

  show(show) {
    DOMUtils.setClass(this._root, show, 'modal--visible');
    document.body.style.overflow = show ? "hidden" : "visible";
  }
```

```
  isVisible() {
    return DOMUtils.hasClass(this._root, 'modal--visible');
  }

  remove() {
    if (this.#wrappedComponent !== undefined) {
      this.#wrappedComponent.remove();
    }
    super.remove();
  }

  set ignoreElements(elements) {
    this.#ignoreElements = elements;
  }
}

Application.registerComponent(Modal);
```

Header

```
import Application from '../application';
import Component from './component';
import DOMUtils from '../utils/dom-utils';

export default class Header extends Component {
  #authButton;
  #catalogButton;
  #cartButton;
  #cartButtonBadge;
  #favoriteButton;
  #favoriteButtonBadge;

  authButtonClickHandler;
  catalogButtonClickHandler;
  cartButtonClickHandler;
  favoriteButtonClickHandler;


  static get componentName() {
    return 'header';
  }

  init() {
    super.init();

    this.#authButton = this._initElement(this._root, '.header__auth-button',
      !Application.isAuthenticated() ? {'click': event =>
this._callOuterHandler(this.authButtonClickHandler, event)} : {}
    );
    this.#catalogButton = this._initElement(this._root, '.header__catalog-
button', {
      'click': event => this._callOuterHandler(this.catalogButtonClickHandler,
event)
    });
    this.#cartButton = this._initElement(this._root, '.header__cart-button', {
      'click': event => this._callOuterHandler(this.cartButtonClickHandler,
event)
    });
    this.#cartButtonBadge = this._initElement(this.#cartButton,
'.header__button-badge');

    this.#favoriteButton = this._initElement(this._root, '.header__favorite-
button', {
```

```
      'click': event => this._callOuterHandler(this.favoriteButtonClickHandler,
event)
    });
    this.#favoriteButtonBadge = this._initElement(this.#favoriteButton,
'.header__button-badge');
  }

  get cartItemCount() {
    return +this.#cartButtonBadge.textContent;
  }

  get favoriteItemCount() {
    return +this.#favoriteButtonBadge.textContent;
  }

  set cartItemCount(count) {
    DOMUtils.setClass(this.#cartButtonBadge, count > 0, 'header__button-badge--
visible');
    this.#cartButtonBadge.textContent = count;
  }

  set favoriteItemCount(count) {
    DOMUtils.setClass(this.#favoriteButtonBadge, count > 0, 'header__button-
badge--visible');
    this.#favoriteButtonBadge.textContent = count;
  }
}

Application.registerComponent(Header);
```

ModelForm

```
import Application from '../../../application';
import Form from '../form';
import DOMUtils from '../../../utils/dom-utils';
import HandlerUtils from '../../../utils/handler-utils';
import ValidationUtils from '../../../utils/validation-utils';

export default class ModelForm extends Form {
  static get componentName() {
    return 'model-form';
  }

  get modeName() {
    if (window.location.pathname.endsWith('/edit')) {
      return 'editing';
    }
    return 'creating';
  }

  _definePhoneControl(name, required) {
    const phoneControl = this._defineControl('tel', name, e => e.value, '', {
      'predicate': value => (!required && value.length === 0) ||
ValidationUtils.PHONE_REGEX.test(value),
      'messageIfInvalid': 'Номер телефону не відповідає формату.',
    });
    this._defineHandlers(phoneControl, HandlerUtils.phoneNumber());
    return phoneControl;
  }

  _defineTextControl(type, name, required, maxLength) {
    return this._defineControl(type, name, e => e.value, '', {
```

```
      'predicate': value => (!required && value.length === 0) || (value.length >
0 && value.length <= maxLength),
      'messageIfInvalid': 'Поле є обов\'язковим для заповнення.',
    });
  }

  _definePhotoControl(name, required, maxSize = 1048576) {
    const control = this._defineControl('file', name, e => e.files, undefined, {
      'predicate': files => (!required && files.length === 0) || files[0].size
<= maxSize,
      'messageIfInvalid': `Зображення надто велике — більше ${maxSize /
1048576}МБ.`,
    });
    const label = this._initElement(this._root, `input[name="${name}"]+
.form__label`);
    this._defineHandlers(control, {
      'change': event => {
        if (this._isValid(event.currentTarget)) {
          const fileReader = new FileReader();
          fileReader.readAsDataURL(event.currentTarget.files[0]);
          fileReader.onload = () => {
            DOMUtils.addClass(label, 'form__label--with-image');
            label.style.backgroundImage = `url("${fileReader.result}")`;
          };
        } else {
          label.removeAttribute('style');
          DOMUtils.removeClass(label, 'form__label--with-image');
        }
      }
    });

    return {
      'control': control,
      'label': label,
    };
  }
}

Application.registerComponent(ModelForm);
```

CategoryForm

```
import ModelForm from './model-form';
import Application from '../../../application';

export default class CategoryForm extends ModelForm {
  static get componentName() {
    return 'category-form';
  }

  init() {
    super.init();

    this._definePhotoControl('photo');
    this._definePhotoControl('icon');
    this._defineTextControl('text', 'name', true, 32);
    this._defineTextControl('textarea', 'description', false, 255);
  }
}

Application.registerComponent(CategoryForm);
```

Додаток Б. Основні контролери клієнтської частини веб-додатку

UserController

```
import Controller from '../controller';
import Application from '../../application';
import Header from '../../components/header';
import Modal from '../../components/modal';
import AuthForm from '../../components/forms/auth-form';
import Alert from '../../components/alert';
import Catalog from '../../components/catalog';
import DOMUtils from '../../utils/dom-utils';
import CookiePolicy from '../../components/cookie-policy';
import Cart from '../../components/cart';
import Card from '../../components/card';
import GuestFavorites from '../../components/guest-favorites';
import SearchBox from '../../components/search-box';
import '../../components/footer';

export default class UserController extends Controller {
  _modal = Application.lookupComponentInstance(Modal);
  _alert = Application.lookupComponentInstance(Alert);
  _cookiePolicy = Application.lookupComponentInstance(CookiePolicy);

  _header = Application.lookupComponentInstance(Header);
  _authForm = Application.lookupComponentInstance(AuthForm);
  _catalog = Application.lookupComponentInstance(Catalog);
  _cart = Application.lookupComponentInstance(Cart);
  _guestFavorites = Application.lookupComponentInstance(GuestFavorites);
  _searchBox = Application.lookupComponentInstance(SearchBox);

  _cards = Application.lookupComponentInstances(Card);
  _cardsMap = new Map();

  _alertRaised = false;
  _navigationNeeded = false;

  #navigateIfNeeded() {
    if (this._navigationNeeded) {
      this._navigationNeeded = false;
      this._modal.wrapComponent(this._authForm);
      this._authForm.switchTo(serverData['navigation']['destination']);
      this._modal.show(true);
    }
  }

  attach() {
    const serverData = Application.getServerData();
    if (serverData) {
      this._navigationNeeded = serverData['navigation'] !== undefined;

      if (serverData['alert']) {
        this._alertRaised = true;
        this._modal.ignoreElements =
[this._alert.root.querySelector('.alert__button')];

        this._modal.wrapComponent(this._alert);
        this._modal.closeButtonClickHandler =
          this._modal.closeHandler =
            this._modal.escapeHandler =
              this._alert.okButtonClickHandler =
```

```
                this._cart.continueButtonClickHandler =
                  event => {
                    this._modal.show(false);
                    this._modal.unwrapComponent();
                    if (this._alertRaised) {
                      this._alertRaised = false;
                      this.#navigateIfNeeded();
                    }
                  }
        this._modal.show(true);
      } else {
        this._modal.wrapComponent(this._authForm);
        this._modal.closeButtonClickHandler =
          this._modal.closeHandler =
            this._modal.escapeHandler =
              this._cart.continueButtonClickHandler =
                event => {
                  this._modal.show(false);
                  this._modal.unwrapComponent();
                }
        this.#navigateIfNeeded();
      }
    }

    // Header
    this._header.authButtonClickHandler = event => {
      this._modal.wrapComponent(this._authForm);
      this._modal.show(true);
    };
    this._header.cartButtonClickHandler = event => {
      this._modal.wrapComponent(this._cart);
      this._modal.show(true);
    }
    this._header.cartItemCount = this._cart.quantity;

    // Catalog
    const catalogButton = this._header.root.querySelector('.header__catalog-
button');
    this._catalog.ignoreElements = [catalogButton];
    this._catalog.closeHandler = () => {
      DOMUtils.removeClass(catalogButton, 'header__catalog-button--open');
    };
    this._header.catalogButtonClickHandler = event => {
      if (this._catalog.isVisible()) {
        DOMUtils.removeClass(event.currentTarget, 'header__catalog-button--
open');
        this._catalog.show(false);
      } else {
        DOMUtils.addClass(event.currentTarget, 'header__catalog-button--open');
        this._catalog.show(true);
      }
    }

    // Cards
    for (const card of this._cards) {
      if (!this._cardsMap.has(card.productID)) {
        this._cardsMap.set(card.productID, new Set());
      }
      this._cardsMap.get(card.productID).add(card);
      if (this._cart.hasItem(card.productID)) {
        card.inCart = true;
      }
```

```
    card.buyButtonClickHandler = (event, productID) => {
      if (!card.isInCart()) {
        this._cart.addItem(productID);
        card.inCart = true;
      }
      this._modal.wrapComponent(this._cart);
      this._modal.show(true);
    }
  }
  this._cart.removeItemHandler = productID => {
    if (this._cardsMap.has(productID)) {
      for (const card of this._cardsMap.get(productID)) {
        card.inCart = false;
      }
    }
  }
  this._cart.updateHandler = quantity => {
    this._header.cartItemCount = quantity;
  }

  // Favorites
  if (!Application.isAuthenticated()) {
    this._header.favoriteButtonClickHandler = event => {
      this._modal.wrapComponent(this._guestFavorites);
      this._modal.show(true);
    }
  }
  this._guestFavorites.continueButtonClickHandler = event => {
    this._modal.unwrapComponent();
    this._modal.show(false);
  }
  this._guestFavorites.saveButtonClickHandler = event => {
    this._authForm.switchTo('signup-form');
    this._modal.wrapComponent(this._authForm);
    this._modal.show(true);
  }

  for (const card of this._cards) {
    if (this._guestFavorites.hasItem(card.productID)) {
      card.inFavorites = true;
    }

    card.favoriteButtonClickHandler = (event, productID) => {
      card.inFavorites = !card.isInFavorites();
      if (card.isInFavorites()) {
        this._guestFavorites.addItem(productID);
      } else {
        this._guestFavorites.removeItem(productID);
      }
    }
  }
  this._guestFavorites.removeItemHandler = productID => {
    if (this._cardsMap.has(productID)) {
      for (const card of this._cardsMap.get(productID)) {
        card.inFavorites = false;
      }
    }
  }
  this._guestFavorites.updateHandler = quantity => {
    this._header.favoriteItemCount = quantity;
  }
}
```

```
  static get controllerName() {
    return 'user-controller';
  }
}

Application.registerController(UserController);
```

AdminController

```
import Controller from '../controller';
import Application from '../../application';
import AdminSidebar from '../../components/admin-sidebar';
import AdminHeader from '../../components/admin-header';
import Alert from '../../components/alert';
import Modal from '../../components/modal';

export default class AdminController extends Controller {
  _adminHeader = Application.lookupComponentInstance(AdminHeader);
  _adminSidebar = Application.lookupComponentInstance(AdminSidebar);
  _modal = Application.lookupComponentInstance(Modal);
  _alert = Application.lookupComponentInstance(Alert);
  _alertRaised = false;

  attach() {
    const serverData = Application.getServerData();
    if (serverData) {
      this._navigationNeeded = serverData['navigation'] !== undefined;

      if (serverData['alert']) {
        this._alertRaised = true;
        this._modal.ignoreElements =
[this._alert.root.querySelector('.alert__button')];

        this._modal.wrapComponent(this._alert);
        this._modal.closeButtonClickHandler =
          this._modal.closeHandler =
            this._modal.escapeHandler =
              this._alert.okButtonClickHandler =
                  event => {
                    this._modal.show(false);
                    this._modal.unwrapComponent();
                    if (this._alertRaised) {
                      this._alertRaised = false;
                    }
                  }
        this._modal.show(true);
      } else {
        this._modal.wrapComponent(this._authForm);
        this._modal.closeButtonClickHandler =
          this._modal.closeHandler =
            this._modal.escapeHandler =
                event => {
                  this._modal.show(false);
                  this._modal.unwrapComponent();
                }
      }
    }

    this._adminHeader.burgerButtonClickHandler = event => {
      if (this._adminSidebar.isVisible()) {
        this._adminSidebar.show(false);
      } else {
        this._adminSidebar.show(true);
```

```
      }
    }
  }

  static get controllerName() {
    return 'admin-controller';
  }
}

Application.registerController(AdminController);
```

AdminTableController

```
import Application from '../../application';
import AdminController from './admin-controller';
import AdminControls from '../../components/admin-controls';
import AdminTable from '../../components/admin-table';

export default class AdminTableController extends AdminController {
  _adminControls = Application.lookupComponentInstance(AdminControls);
  _adminTable = Application.lookupComponentInstance(AdminTable);

  attach() {
    super.attach();
    this._adminControls.columnCheckboxesChangeHandler = (number, show) => {
      this._adminTable.showColumnByNumber(number, show);
    }

    let number = 0;
    for (const state of this._adminControls.statesOfColumnCheckboxes) {
      this._adminTable.showColumnByNumber(++number, state);
    }
  }

  static get controllerName() {
    return 'admin-table-controller';
  }
}

Application.registerController(AdminTableController);
```

Додаток В. Точки входу

User

```
import Application from '../application';

// Контролери для сторінок користувача
import '../controllers/user/home-controller';
import '../controllers/user/shops-controller';
import '../controllers/user/checkout-controller';
import '../controllers/user/product-controller';
import '../controllers/user/profile-controller';
import '../controllers/user/orders-controller';
import '../controllers/user/catalog-type-controller';

Application.init();
```

Admin

```
import Application from '../application.js';

// Контролери для сторінок адміністратора
import '../controllers/admin/admin-controller';
import '../controllers/admin/dashboard-controller';
import '../controllers/admin/admin-table-controller';
import '../controllers/admin/order-table-controller';
import '../controllers/admin/model-form-controller';

Application.init();
```

Додаток Г. Лістинги міграцій

2022_04_30_000000_create_roles_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('roles', function (Blueprint $table) {
            $table->unsignedTinyInteger('id')->autoIncrement();
            $table->string('name')->unique();
            $table->string('title')->unique();
        });
    }

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

2022_04_30_000001_create_users_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->unsignedInteger('id')->autoIncrement();
            $table->unsignedTinyInteger('role_id');
            $table->foreign('role_id')->references('id')->on('roles');
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password', 60);
            $table->string('phone_number', 50)->nullable();
            $table->rememberToken();
            $table->timestamps();
            $table->softDeletes();
        });
    }

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

2022_04_30_000002_create_categories_table.php

```
<?php
```

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

return new class extends Migration
{
    public function up()
    {
        Schema::create('categories', function (Blueprint $table) {
            $table->unsignedTinyInteger('id')->autoIncrement();
            $table->string('name', 32)->unique();
            $table->text('description')->nullable();
            $table->string('photo')->nullable();
            $table->string('icon')->nullable();
            $table->timestamps();
            $table->softDeletes();
        });
    }

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

2022_04_30_000003_create_types_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('types', function (Blueprint $table) {
            $table->unsignedSmallInteger('id')->autoIncrement();
            $table->unsignedTinyInteger('category_id');
            $table->foreign('category_id')->references('id')->on('categories');
            $table->string('name', 64);
            $table->text('description')->nullable();
            $table->timestamps();
            $table->softDeletes();
            $table->unique(['category_id', 'name']);
        });
    }

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

2022_04_30_000004_create_vendors_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
```

```
{
    public function up()
    {
        Schema::create('vendors', function (Blueprint $table) {
            $table->unsignedSmallInteger('id')->autoIncrement();
            $table->string('name', 32)->unique();
            $table->string('logo')->nullable();
            $table->timestamps();
            $table->softDeletes();
        });
    }

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

2022_04_30_000005_create_products_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('products', function (Blueprint $table) {
            $table->unsignedInteger('id')->autoIncrement();
            $table->unsignedSmallInteger('vendor_id');
            $table->foreign('vendor_id')->references('id')->on('vendors');
            $table->string('SKU', 32)->unique();
            $table->string('name');
            $table->text('description')->nullable();
            $table->unsignedFloat('price');
            $table->string('preview')->nullable();
            $table->string('photo')->nullable();
            $table->timestamps();
            $table->softDeletes();
        });
    }

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

2022_04_30_000006_create_product_types_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('product_types', function (Blueprint $table) {
```

```
            $table->unsignedInteger('id')->autoIncrement();
            $table->unsignedInteger('product_id');
            $table->foreign('product_id')->references('id')->on('products');
            $table->unsignedSmallInteger('type_id');
            $table->foreign('type_id')->references('id')->on('types');
            $table->unique(['product_id', 'type_id']);
        });
    }

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

2022_04_30_000007_create_shops_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('shops', function (Blueprint $table) {
            $table->unsignedTinyInteger('id')->autoIncrement();
            $table->string('address')->unique();
            $table->string('email');
            $table->string('phone_number', 50);
            $table->string('working_hours');
            $table->string('photo')->nullable();
            $table->double('latitude');
            $table->double('longitude');
            $table->timestamps();
            $table->softDeletes();
        });
    }

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

2022_04_30_000008_create_favorites_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('favorites', function (Blueprint $table) {
            $table->unsignedInteger('id')->autoIncrement();
            $table->unsignedInteger('user_id');
            $table->foreign('user_id')->references('id')->on('users');
            $table->unsignedInteger('product_id');
```

```
            $table->foreign('product_id')->references('id')->on('products');
            $table->timestampTz('created_at')->nullable();
            $table->unique(['user_id', 'product_id']);
        });
    }

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

2022_04_30_000009_create_feedback_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('feedback', function (Blueprint $table) {
            $table->unsignedInteger('id')->autoIncrement();
            $table->unsignedInteger('user_id');
            $table->foreign('user_id')->references('id')->on('users');
            $table->string('message');
            $table->timestampTz('created_at')->nullable();
        });
    }

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

2022_04_30_000010_create_cart_items_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('cart_items', function (Blueprint $table) {
            $table->unsignedInteger('id')->autoIncrement();
            $table->unsignedInteger('user_id');
            $table->foreign('user_id')->references('id')->on('users');
            $table->unsignedInteger('product_id');
            $table->foreign('product_id')->references('id')->on('products');
            $table->unsignedTinyInteger('quantity');
            $table->unique(['user_id', 'product_id']);
        });
    }

    public function down()
    {
```

```
        Schema::dropIfExists('cart_items');
    }
};
```

2022_04_30_000011_create_order_statuses_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('order_statuses', function (Blueprint $table) {
            $table->unsignedTinyInteger('id')->autoIncrement();
            $table->string('name', 32)->unique();
            $table->string('color', 7)->unique();
        });
    }

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

2022_04_30_000012_create_orders_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('orders', function (Blueprint $table) {
            $table->unsignedBigInteger('id')->autoIncrement();
            $table->unsignedTinyInteger('shop_id');
            $table->foreign('shop_id')->references('id')->on('shops');
            $table->unsignedInteger('user_id')->nullable();
            $table->foreign('user_id')->references('id')->on('users');
            $table->string('name');
            $table->string('phone_number', 50);
            $table->boolean('without_callback');
            $table->string('email');
            $table->string('comment')->nullable();
            $table->unsignedTinyInteger('order_status_id');
            $table->foreign('order_status_id')->references('id')-
>on('order_statuses');
            $table->timestamps();
        });
    }

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

2022_04_30_000013_create_order_products_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('order_products', function (Blueprint $table) {
            $table->unsignedBigInteger('id')->autoIncrement();
            $table->unsignedBigInteger('order_id');
            $table->foreign('order_id')->references('id')->on('orders');
            $table->unsignedInteger('product_id');
            $table->foreign('product_id')->references('id')->on('products');
            $table->unsignedFloat('price');
            $table->unsignedTinyInteger('quantity');
            $table->unique(['order_id', 'product_id']);
        });
    }

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

2022_04_30_000014_create_slide_types_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('slide_types', function (Blueprint $table) {
            $table->unsignedTinyInteger('id')->autoIncrement();
            $table->string('name', 32)->unique();
        });
    }

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

2022_04_30_000015_create_slides_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
```

```
    public function up()
    {
        Schema::create('slides', function (Blueprint $table) {
            $table->unsignedInteger('id')->autoIncrement();
            $table->unsignedTinyInteger('slide_type_id');
            $table->foreign('slide_type_id')->references('id')-
>on('slide_types');
            $table->unsignedInteger('link');
            $table->string('slide')->nullable();
        });
    }

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

2022_04_30_000016_create_settings_table.php

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up()
    {
        Schema::create('settings', function (Blueprint $table) {
            $table->unsignedTinyInteger('id')->autoIncrement();
            $table->string('name');
            $table->string('value')->nullable();
        });
    }

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

Додаток Д. Лістинги основних контролерів серверної частини

AccountController

```
<?php

namespace App\Http\Controllers;

use App\Models\Favorite;
use App\Models\Order;
use App\Models\User;
use App\Rules\Password;
use App\Rules\PhoneNumber;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Validator;

class AccountController extends Controller
{
    public function profile(Request $request)
    {
        return view('sections.account.profile');
    }

    public function update(Request $request) {
        $validator = Validator::make($request->all(), [
            'name' => [
                'required',
                'string',
                'max:255',
            ],
            'email' => [
                'required',
                'string',
                'email',
                'max:255',
            ],
            'phone-number' => [
                'nullable',
                'string',
                new PhoneNumber(),
            ],
            'password' => [
                'nullable',
                'string',
            ],
            'new-password' => [
                'nullable',
                'string',
                new Password(),
            ],
        ]);

        $validator->after(function ($validator) use ($request) {
            if (mb_strlen($request['new-password']) !== 0) {
                if (mb_strlen($request['password']) === 0) {
                    $validator->errors()->add('password', 'Треба ввести
пароль.');
                }
            }
            if (Auth::user()->email !== $request['email'] &&
```

```
                User::where('email', $request['email'])->count() > 0) {
                $validator->errors()->add('email', 'Електронна пошта вже
використовується.');
            }
        });

        if ($validator->fails()) {
            return redirect_back()
                ->withInput()
                ->withErrors($validator->getMessageBag(), 'profile')
                ->with([
                    'alert' => [
                        'type' => 'error',
                        'heading' => 'Особисті дані',
                        'message' => 'Дані не відповідають вимогам.',
                    ],
                ]);
        }

        $newUserData = [
            'name' => $request['name'],
            'email' => $request['email'],
            'phone_number' => $request['phone-number'],
        ];
        if (mb_strlen($request['new-password']) > 0) {
            $newUserData['password'] = bcrypt($request['new-password']);
        }
        Auth::user()->update($newUserData);

        return redirect()->route('account.profile')->with([
            'alert' => [
                'type' => 'success',
                'heading' => 'Особисті дані',
                'message' => 'Дані були успішно оновлені.',
            ],
        ]);
    }

    public function orders(Request $request)
    {
        return pagination(
            $request,
            'account.orders',
            fn() => Order::with('orderStatus', 'orderProducts',
'orderProducts.product', 'orderProducts.product.productTypes', 'shop')-
>where('user_id', Auth::id())->get(),
            'orders',
            'sections.account.orders',
            [],
        );
    }

    public function favorites(Request $request)
    {
        return pagination(
            $request,
            'account.favorites',
            fn() => Favorite::with('product', 'product.productTypes')-
>where('user_id', Auth::id())->latest()->get(),
            'favorites',
            'sections.account.favorites',
            []
        );
```

```
    }
}
```

AuthController

```
<?php

namespace App\Http\Controllers;

use App\Models\CartItem;
use App\Models\Favorite;
use App\Models\Product;
use App\Models\Role;
use App\Models\User;
use App\Rules\Password;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Session;
use Illuminate\Support\Facades\Validator;

class AuthController extends Controller
{
    public function signUp(Request $request) {
        $validator = Validator::make($request->all(), [
            'name' => [
                'required',
                'string',
                'max:255',
            ],
            'email' => [
                'required',
                'string',
                'unique:users,email',
                'email',
                'max:255',
            ],
            'password' => [
                'required',
                'string',
                new Password(),
            ],
        ]);

        if ($validator->fails()) {
            return redirect_back()
                ->withInput()
                ->withErrors($validator->getMessageBag(), 'signup')
                ->with([
                    'alert' => [
                        'type' => 'error',
                        'heading' => 'Реєстрація',
                        'message' => 'Дані не відповідають вимогам.',
                    ],
                    'navigation' => [
                        'destination' => 'signup-form',
                    ],
                ]);
        }

        $user = new User([
            'role_id' => Role::where('name', 'user')->value('id'),
            'name' => $request['name'],
            'email' => $request['email'],
```

```
            'password' => bcrypt($request['password']),
        ]);
        $user->save();

        Auth::login($user);

        $cartItems = Session::get('cartItems');
        foreach ($cartItems as $productID => $cartItem) {
            (new CartItem([
                'user_id' => $user->id,
                'product_id' => $productID,
                'quantity' => $cartItem['quantity'],
            ]))->save();
        }

        $favorites = Session::get('favorites');
        foreach ($favorites as $productID => $favorite) {
            (new Favorite([
                'user_id' => $user->id,
                'product_id' => $productID,
            ]))->save();
        }

        return redirect_back()->with([
            'alert' => [
                'type' => 'success',
                'heading' => 'Реєстрація',
                'message' => 'Акаунт був успішно створений.',
            ],
        ]);
    }

    public function logIn(Request $request) {
        $validator = Validator::make($request->all(),[
            'email' => [
                'required',
                'string',
                'email',
                'max:255',
            ],
            'password' => [
                'required',
                'string',
            ],
            'remember-me' => [
                'sometimes',
                'accepted',
            ],
        ]);

        if ($validator->fails()) {
            return redirect_back()
                ->withInput()
                ->withErrors($validator->getMessageBag(), 'login')
                ->with([
                    'alert' => [
                        'type' => 'error',
                        'heading' => 'Вхід',
                        'message' => 'Дані не відповідають вимогам.',
                    ],
                    'navigation' => [
                        'destination' => 'login-form',
                    ],
```

```
                ]);
        }

        if (Auth::attempt([
            'email' => $request['email'],
            'password' => $request['password'],
        ], $request->has('remember-me'))) {
            $cartItems = Session::get('cartItems');
            $userID = Auth::id();
            $storedCartItems = CartItem::where('user_id', $userID)->get()-
>keyBy('product_id');
            $storedCartItemsArray = $storedCartItems->toArray();

            foreach ($cartItems as $productID => $cartItem) {
                if (!array_key_exists($productID, $storedCartItemsArray)) {
                    (new CartItem([
                        'user_id' => $userID,
                        'product_id' => $productID,
                        'quantity' => $cartItem['quantity'],
                    ]))->save();
                } else if ($cartItem['quantity'] > $storedCartItems[$productID]-
>quantity) {
                    CartItem
                        ::where('user_id', $userID)
                        ->where('product_id', $productID)
                        ->update([
                            'quantity' => $cartItem['quantity'],
                        ]);
                }
            }

            foreach ($storedCartItems as $storedCartItem) {
                if (!array_key_exists($storedCartItem->product_id, $cartItems))
{
                    $cartItems[$storedCartItem->product_id] = [
                        'quantity' => $storedCartItem->quantity,
                    ];
                } else if ($cartItems[$storedCartItem->product_id]['quantity'] <
$storedCartItem->quantity) {
                    $cartItems[$storedCartItem->product_id]['quantity'] =
$storedCartItem->quantity;
                }
            }
            Session::put('cartItems', $cartItems);

            $favorites = Session::get('favorites');
            $storedFavorites = Favorite::where('user_id', $userID)->get()-
>keyBy('product_id');
            $storedFavoritesArray = $storedFavorites->toArray();

            foreach ($favorites as $productID => $favorite) {
                if (!array_key_exists($productID, $storedFavoritesArray)) {
                    (new Favorite([
                        'user_id' => $userID,
                        'product_id' => $productID,
                    ]))->save();
                }
            }

            foreach ($storedFavorites as $storedFavorite) {
                if (!array_key_exists($storedFavorite->product_id, $favorites))
{
                    $favorites[$storedFavorite->product_id] = [];
```

```
                }
            }
            Session::put('favorites', $favorites);

            return redirect_back();
        }

        return redirect_back()->with([
            'alert' => [
                'type' => 'error',
                'heading' => 'Вхід',
                'message' => 'Не вдалося авторизуватися на сайті.',
            ],
            'navigation' => [
                'destination' => 'login-form',
            ],
        ]);
    }

    public function logOut(Request $request) {
        Auth::logout();
        return redirect_back();
    }

    public function checkEmailExistence(Request $request)
    {
        $request->validate([
            'email' => [
                'required',
                'string',
                'email',
            ],
        ]);

        return [
            'exists' => User::where('email', $request['email'])->exists(),
        ];
    }
}
```

PrimaryController

```
<?php

namespace App\Http\Controllers;

use App\Models\Category;
use App\Models\OrderProduct;
use App\Models\Product;
use App\Models\Shop;
use App\Models\Slide;
use App\Models\Type;
use App\Models\Vendor;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Str;

class PrimaryController extends Controller
{
    public function home(Request $request)
    {
        $slides = Slide::with('slideType')->whereNotNull('slide')->get();
        foreach ($slides as $slide) {
```

```
            switch ($slide->slideType->name) {
                case 'Продукт':
                    $product = Product::with('productTypes')->find($slide-
>link);
                    $slide['href'] = route('catalog.product', ['productID' =>
$slide->link, 'slug' => Str::slug($product->name, '-')]) . '?t=' . $product-
>productTypes[0]->type_id;
                    break;
                case 'Тип':
                    $slide['href'] = route('catalog.type', ['typeID' => $slide-
>link, 'slug' => Str::slug(Type::find($slide->link)->name)]);
                    break;
                case 'Категорія':
                    $slide['href'] = route('catalog.category', ['categoryID' =>
$slide->link, 'slug' => Str::slug(Category::find($slide->link)->name)]);
                    break;
                case 'Постачальник':
                    $slide['href'] = route('catalog.vendor', ['vendorID' =>
$slide->link, 'slug' => Str::slug(Vendor::find($slide->link)->name)]);
                    break;
            }
        }

        return view('sections.static.home', [
            'slides' => $slides,
            'offer' => [
                'new' => [
                    'title' => 'Новинки',
                    'products' => Product
                        ::with('productTypes')
                        ->orderBy('created_at', 'DESC')
                        ->limit(get_setting('LATEST_SIZE'))
                        ->get(),
                ],
                'hit' => [
                    'title' => 'Хіти продажів',
                    'products' => Product
                        ::with('productTypes')
                        ->whereIn('id', function ($query) {
                            return $query->fromSub(function ($subQuery) {
                                $subQuery
                                    ->select('product_id')
                                    ->from(with(new OrderProduct())->getTable())
                                    ->groupBy('product_id')
                                    ->orderBy(DB::raw('SUM(quantity)'), 'DESC')
                                    ->limit(get_setting('HITS_SIZE'));
                            }, 'sq');
                        })
                        ->get(),
                ],
            ],
        ]);
    }

    public function about(Request $request)
    {
        return view('sections.static.about', [
            'breadcrumbs' => [
                ['name' => 'Головна', 'link' => route('home')],
                ['name' => 'Про нас'],
            ],
        ]);
    }
```

```
    public function paymentAndDelivery(Request $request)
    {
        return view('sections.static.payment-and-delivery', [
            'breadcrumbs' => [
                ['name' => 'Головна', 'link' => route('home')],
                ['name' => 'Оплата та доставка'],
            ]
        ]);
    }

    public function exchangeAndReturn(Request $request)
    {
        return view('sections.static.exchange-and-return', [
            'breadcrumbs' => [
                ['name' => 'Головна', 'link' => route('home')],
                ['name' => 'Обмін та повернення'],
            ]
        ]);
    }

    public function contacts(Request $request)
    {
        return view('sections.static.contacts', [
            'breadcrumbs' => [
                ['name' => 'Головна', 'link' => route('home')],
                ['name' => 'Контакти'],
            ]
        ]);
    }

    public function shops(Request $request)
    {
        return view('sections.static.shops', [
            'breadcrumbs' => [
                ['name' => 'Головна', 'link' => route('home')],
                ['name' => 'Магазини'],
            ],
            'shops' => Shop::all(),
        ]);
    }
}
```

CatalogController

```
<?php

namespace App\Http\Controllers;

use App\Models\Category;
use App\Models\Product;
use App\Models\ProductType;
use App\Models\Shop;
use App\Models\Type;
use App\Models\Vendor;
use Illuminate\Database\Eloquent\ModelNotFoundException;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Facades\Session;
use Illuminate\Support\Str;
use Illuminate\View\ComponentAttributeBag;

class CatalogController extends Controller
```

```
{
    private static $sortMappings = [
        'cheap' => ['price', 'asc'],
        'expensive' => ['price', 'desc'],
        null => ['price', 'asc'],
    ];

    private static $sortConfiguration = [
        'sortOptions' => [
            'Від дешевих до дорогих' => 'cheap',
            'Від дорогих до дешевих' => 'expensive',
        ],
        'defaultSort' => 'cheap',
    ];

    private static function getPriceFilter($requestFilters)
    {
        $minPrice = floor(Product::min('price'));
        $maxPrice = ceil(Product::max('price'));
        return [
            'heading' => 'Вартість',
            'type' => 'range',
            'min' => $minPrice,
            'max' => $maxPrice,
            'step' => 1,
            'from' => $requestFilters['price']['from'] ?? $minPrice,
            'to' => $requestFilters['price']['to'] ?? $maxPrice,
        ];
    }

    private static function getVendorFilter($requestFilters, $vendors)
    {
        $vendorValues = $requestFilters['vendor']['values'] ?? [];
        return [
            'heading' => 'Постачальник',
            'type' => 'list',
            'column' => 'vendor_id',
            'skip' => count($vendorValues) === 0,
            'values' => $vendorValues,
            'vendors' => $vendors,
        ];
    }

    public function category($slug, $categoryID)
    {
        $category = Category::findOrFail($categoryID);
        return view('sections.catalog.category', [
            'category' => $category,
            'breadcrumbs' => [
                ['name' => 'Головна', 'link' => route('home')],
                ['name' => $category->name],
            ],
            'types' => DB::select(
                <<<SQL
select types.id as type_id, types.name as type_name, p.id as product_id,
p.preview as product_preview from types
join products as p on p.id = (
  select p2.id from products as p2
  join product_types on p2.id = product_types.product_id
  where product_types.type_id = types.id
    and p2.preview is not null
  limit 1
)
```

```
where types.category_id = $categoryID
SQL
            ),
        ]);
    }

    public function type($slug, $typeID, Request $request)
    {
        $type = Type::findOrFail($typeID);
        $category = $type->category;
        $requestFilters = get_filters($request);
        $filters = [
            'price' => self::getPriceFilter($requestFilters),
            'vendor' => self::getVendorFilter($requestFilters, Vendor
                ::select('vendors.*')
                ->join('products', 'products.vendor_id', '=', 'vendors.id')
                ->join('product_types', 'products.id', '=',
'product_types.product_id')
                ->where('product_types.type_id', $typeID)
                ->get()),
        ];
        return pagination(
            $request,
            route('home'),
            fn() => filter(Product
                ::join('product_types', 'products.id', '=',
'product_types.product_id')
                ->select('products.*')
                ->where('product_types.type_id', $typeID)
                ->orderBy(...self::$sortMappings[$request['sort'] ?? null]),
$filters)
                ->get(),
            'products',
            'sections.catalog.type',
            array_merge([
                'type' => $type,
                'breadcrumbs' => [
                    ['name' => 'Головна', 'link' => route('home')],
                    ['name' => $category->name, 'link' =>
route('catalog.category', ['slug' => Str::slug($category->name, '-'),
'categoryID' => $category->id])],
                    ['name' => $type->name],
                ],
                'filters' => $filters,
            ], self::$sortConfiguration)
        );
    }

    public function product($slug, $productID, Request $request)
    {
        $product = Product::findOrFail($productID);
        $typeID = $request->get('t');

        if (ProductType::where('product_id', $product->id)->where('type_id',
$typeID)->count() === 0) {
            throw new ModelNotFoundException();
        }

        $type = Type::find($typeID);
        $category = Category::find($type->category_id);
        return view('sections.catalog.product', [
            'product' => $product,
            'breadcrumbs' => [
```

```
                ['name' => 'Головна', 'link' => route('home')],
                ['name' => $category->name, 'link' => route('catalog.category',
['slug' => Str::slug($category->name, '-'), 'categoryID' => $category->id])],
                ['name' => $type->name, 'link' => route('catalog.type', ['slug'
=> $slug, 'typeID' => $type->id])],
                ['name' => $product->name],
            ],
        ]);
    }

    public function vendor($slug, $vendorID, Request $request)
    {
        $vendor = Vendor::findOrFail($vendorID);
        $requestFilters = get_filters($request);
        $filters = [
            'price' => self::getPriceFilter($requestFilters),
        ];
        return pagination(
            $request,
            route('home'),
            fn() => filter(
                Product
                    ::where('vendor_id', $vendorID)
                    ->orderBy(...self::$sortMappings[$request['sort'] ?? null]),
$filters)
                ->get(),
            'products',
            'sections.catalog.vendor',
            array_merge([
                'breadcrumbs' => [
                    ['name' => 'Головна', 'link' => route('home')],
                    ['name' => $vendor->name],
                ],
                'vendor' => $vendor,
                'filters' => $filters,
            ], self::$sortConfiguration)
        );
    }

    public function search(Request $request)
    {
        $query = $request['query'];

        if (!$query) {
            return redirect_back()->with([
                'alert' => [
                    'type' => 'error',
                    'heading' => 'Пошук',
                    'message' => 'Пошуковий запит не може бути порожнім.',
                ],
            ]);
        }

        $requestFilters = get_filters($request);
        $filters = [
            'price' => self::getPriceFilter($requestFilters),
            'vendor' => self::getVendorFilter($requestFilters, Vendor::all()),
        ];
        return pagination(
            $request,
            null,
            fn() => filter(Product
                ::where(function ($subquery) use ($query) {
```

```
                    $subquery
                        ->where('SKU', 'like', '%' . $query . '%')
                        ->orWhere('name', 'like', '%' . $query . '%')
                        ->orWhere('description', 'like', '%' . $query . '%');
                })
                ->orderBy(...self::$sortMappings[$request['sort'] ?? null]),
$filters)
                ->get(),
            'products',
            'sections.catalog.search',
            array_merge([
                'breadcrumbs' => [
                    ['name' => 'Головна', 'link' => route('home')],
                    ['name' => "Результати пошуку \"{$query}\""],
                ],
                'filters' => $filters,
            ], self::$sortConfiguration)
        );
    }

    public function searchResults(Request $request)
    {
        $query = $request['query'];
        return view('components.search-box', [
            'attributes' => new ComponentAttributeBag(),
            'query' => $query,
            'products' => Product
                ::where('SKU', 'like', '%' . $query . '%')
                ->orWhere('name', 'like', '%' . $query . '%')
                ->orWhere('description', 'like', '%' . $query . '%')
                ->limit(get_setting('SEARCH_SIZE'))
                ->get(),
        ]);
    }
}
```

FeedbackController

```
<?php

namespace App\Http\Controllers;

use App\Models\Feedback;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Validator;

class FeedbackController extends Controller
{
    public function index(Request $request)
    {
        return pagination(
            $request,
            'admin.feedback.index',
            fn() => Feedback::latest()->get(),
            'feedback',
            'sections.admin.feedback.index',
            []
        );
    }

    public function create(Request $request)
    {
```

```
        return view('sections.static.feedback', [
            'breadcrumbs' => [
                ['name' => 'Головна', 'link' => route('home')],
                ['name' => 'Зворотній зв\'язок'],
            ],
        ]);
    }

    public function store(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'message' => [
                'required',
                'string',
                'max:255',
            ],
        ]);

        if ($validator->fails()) {
            return redirect_back()
                ->withInput()
                ->withErrors($validator->getMessageBag(), 'feedback')
                ->with([
                    'alert' => [
                        'type' => 'error',
                        'heading' => 'Відгук',
                        'message' => 'Уміст відгуку не відповідає вимогам.',
                    ],
                ]);
        }

        (new Feedback([
            'user_id' => Auth::user()->id,
            'message' => $request['message']
        ]))->save();
        return redirect_back()
            ->with([
                'alert' => [
                    'type' => 'success',
                    'heading' => 'Відгук',
                    'message' => 'Відгук був успішно надісланий.',
                ]
            ]);
    }

    public function delete($id, Request $request)
    {
        Feedback::find($id)?->delete();
        return redirect_back();
    }
}
```

CartController

```
<?php

namespace App\Http\Controllers;

use App\Models\CartItem;
use App\Models\Product;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Session;
```

```
use Illuminate\View\ComponentAttributeBag;

class CartController extends Controller
{
    public function items()
    {
        return view('components.cart', [
            'attributes' => new ComponentAttributeBag(),
            'products' => Product::with('productTypes')->whereIn('id',
array_keys(Session::get('cartItems')))->get(),
            'items' => Session::get('cartItems'),
        ])->render();
    }

    public function addItem(Request $request)
    {
        $id = $request['productID'];
        $cartItems = Session::get('cartItems');
        if (!array_key_exists($id, $cartItems)) {
            $cartItems[$id] = [
                'quantity' => 1,
            ];
            Session::put('cartItems', $cartItems);

            if (!Auth::guest()) {
                (new CartItem([
                    'user_id' => Auth::id(),
                    'product_id' => $id,
                    'quantity' => 1,
                ]))->save();
            }
        }
        return $this->items();
    }

    public function removeItem(Request $request)
    {
        $id = $request['productID'];
        $cartItems = Session::get('cartItems');
        if (array_key_exists($id, $cartItems)) {
            unset($cartItems[$id]);
            Session::put('cartItems', $cartItems);

            if (!Auth::guest()) {
                CartItem
                    ::where('user_id', Auth::id())
                    ->where('product_id', $id)
                    ->delete();
            }
        }
        return $this->items();
    }

    public function setItemQuantity(Request $request)
    {
        $id = $request['productID'];
        $cartItems = Session::get('cartItems');
        if (array_key_exists($id, $cartItems)) {
            $cartItems[$id]['quantity'] = $request['quantity'];
            Session::put('cartItems', $cartItems);

            if (!Auth::guest()) {
                CartItem
```

```
                    ::where('user_id', Auth::id())
                    ->where('product_id', $id)
                    ->update([
                        'quantity' => $request['quantity'],
                    ]);
            }
        }
        return $this->items();
    }
}
```

FavoriteController

```
<?php

namespace App\Http\Controllers;

use App\Models\Favorite;
use App\Models\Product;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Session;
use Illuminate\View\ComponentAttributeBag;

class FavoriteController extends Controller
{
    public function items()
    {
        return view('components.guest-favorites', [
            'attributes' => new ComponentAttributeBag(),
            'products' => Product::with('productTypes')->whereIn('id',
array_keys(Session::get('favorites')))->latest()->get(),
            'items' => Session::get('favorites'),
        ])->render();
    }

    public function addItem(Request $request)
    {
        $id = $request['productID'];
        $favorites = Session::get('favorites');
        if (!array_key_exists($id, $favorites)) {
            $favorites[$id] = [];
            Session::put('favorites', $favorites);

            if (!Auth::guest()) {
                (new Favorite([
                    'user_id' => Auth::id(),
                    'product_id' => $id,
                ]))->save();
            }
        }
        return $this->items();
    }

    public function removeItem(Request $request)
    {
        $id = $request['productID'];
        $favorites = Session::get('favorites');
        if (array_key_exists($id, $favorites)) {
            unset($favorites[$id]);
            Session::put('favorites', $favorites);

            if (!Auth::guest()) {
```

```
                Favorite
                    ::where('user_id', Auth::id())
                    ->where('product_id', $id)
                    ->delete();
            }
        }
        return $this->items();
    }
}
```

Додаток Е. Лістинг із визначенням маршрутів

```
<?php

use App\Http\Controllers\AccountController;
use App\Http\Controllers\AdminController;
use App\Http\Controllers\AuthController;
use App\Http\Controllers\CartController;
use App\Http\Controllers\CatalogController;
use App\Http\Controllers\CategoryController;
use App\Http\Controllers\FavoriteController;
use App\Http\Controllers\FeedbackController;
use App\Http\Controllers\OrderController;
use App\Http\Controllers\PrimaryController;
use App\Http\Controllers\ProductController;
use App\Http\Controllers\ShopController;
use App\Http\Controllers\TypeController;
use App\Http\Controllers\UserController;
use App\Http\Controllers\VendorController;
use Illuminate\Support\Facades\Route;

Route::middleware('session.initialized')->group(function () {
    Route::get('/', [PrimaryController::class, 'home'])->name('home');

    Route::get('/about', [PrimaryController::class, 'about'])->name('about');
    Route::get('/payment-and-delivery', [PrimaryController::class,
'paymentAndDelivery'])->name('payment-and-delivery');
    Route::get('/exchange-and-return', [PrimaryController::class,
'exchangeAndReturn'])->name('exchange-and-return');
    Route::get('/contacts', [PrimaryController::class, 'contacts'])-
>name('contacts');
    Route::get('/shops', [PrimaryController::class, 'shops'])->name('shops');
    Route::middleware('auth')->get('/feedback/create',
[FeedbackController::class, 'create'])->name('feedback.create');
    Route::middleware('auth')->post('/feedback', [FeedbackController::class,
'store'])->name('feedback.store');

    Route::get('/checkout', [OrderController::class, 'create'])-
>name('orders.create');
    Route::post('/orders', [OrderController::class, 'store'])-
>name('orders.store');

    Route::get('/promo', [PrimaryController::class, 'promo'])->name('promo');


    Route::group(['prefix' => 'cart', 'as' => '.cart'], function () {
        Route::post('/items', [CartController::class, 'items']);
        Route::post('/add-item', [CartController::class, 'addItem']);
        Route::post('/remove-item', [CartController::class, 'removeItem']);
        Route::post('/set-item-quantity', [CartController::class,
'setItemQuantity']);
    });

    Route::group(['prefix' => 'favorites', 'as' => '.favorites'], function () {
        Route::post('/items', [FavoriteController::class, 'items']);
        Route::post('/add-item', [FavoriteController::class, 'addItem']);
        Route::post('/remove-item', [FavoriteController::class, 'removeItem']);
    });

    Route::post('/account/check-email-existence', [AuthController::class,
'checkEmailExistence']);
```

```
    Route::group(['as' => 'auth.'], function () {
        Route::post('/signup', [AuthController::class, 'signUp'])-
>name('signup');
        Route::post('/login', [AuthController::class, 'logIn'])->name('login');
        Route::middleware('auth')->get('/logout', [AuthController::class,
'logOut'])->name('logout');
    });

    Route::middleware('auth')->group(function () {
        Route::group(['prefix' => 'account', 'as' => 'account.'], function () {
            Route::get('/profile', [AccountController::class, 'profile'])-
>name('profile');
            Route::post('/profile', [AccountController::class, 'update'])-
>name('update');
            Route::get('/orders', [AccountController::class, 'orders'])-
>name('orders');
            Route::get('/favorites', [AccountController::class, 'favorites'])-
>name('favorites');
        });

        Route::middleware('admin')->group(function () {
            Route::group(['prefix' => 'admin', 'as' => 'admin.'], function () {
                Route::get('/dashboard', [AdminController::class, 'dashboard'])-
>name('dashboard');

                Route::resource('users', UserController::class)-
>except(['show']);
                Route::resource('shops', ShopController::class);
                Route::resource('categories', CategoryController::class);
                Route::resource('types', TypeController::class);
                Route::resource('vendors', VendorController::class);
                Route::resource('products', ProductController::class);
                Route::resource('orders', OrderController::class);

                Route::get('/feedback', [FeedbackController::class, 'index'])-
>name('feedback.index');
                Route::delete('/feedback/{feedbackID}',
[FeedbackController::class, 'delete'])->name('feedback.delete');
            });
        });
    });

    Route::group(['as' => 'catalog.'], function () {
        Route::get('/{slug}/c{categoryID}', [CatalogController::class,
'category'])->name('category');
        Route::get('/{slug}/t{typeID}', [CatalogController::class, 'type'])-
>name('type');
        Route::get('/{slug}/p{productID}', [CatalogController::class,
'product'])->name('product');
        Route::get('/{slug}/v{vendorID}', [CatalogController::class, 'vendor'])-
>name('vendor');
        Route::get('/search', [CatalogController::class, 'search'])-
>name('search');
        Route::get('/search-results', [CatalogController::class,
'searchResults'])->name('search-results');
    });
});
```

Додаток Ж. Допоміжні функції для серверної частини

```
<?php

use App\Models\Setting;
use Illuminate\Database\Eloquent\Builder;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\File;
use Illuminate\Support\Facades\Log;

function component($componentName, $attributes)
{
    $outerComponentName = $attributes->get('class');
    return $outerComponentName ? "{$outerComponentName} {$componentName}" :
$componentName;
}

function wrap_model_data($modelData, $attributes)
{
    $wrapper = [];
    foreach ($attributes as $attribute) {
        if (isset($modelData[$attribute])) {
            $wrapper[$attribute] = $modelData[$attribute];
        }
    }
    return $wrapper;
}

function price_format($price)
{
    return str_replace('.00', '', number_format($price, 2));
}

function validate_pagination_before_query($currentPage, $pageSize, $maxPageSize)
{
    $alert = [
        'type' => 'error',
        'heading' => 'Пагінація',
    ];

    if (!is_int($currentPage)) {
        $alert['message'] = 'Номер сторінки повинен бути цілим числом.';
        return $alert;
    }

    if (!is_int($pageSize)) {
        $alert['message'] = 'Розмір сторінки повинен бути цілим числом.';
        return $alert;
    }

    if ($currentPage <= 0) {
        $alert['message'] = 'Номер сторінки повинен бути додатнім числом.';
        return $alert;
    }

    if ($pageSize <= 0) {
        $alert['message'] = 'Розмір сторінки повинен бути додатнім числом.';
        return $alert;
    }
```

```
    if ($pageSize > $maxPageSize) {
        $alert['message'] = "Максимальний розмір сторінки рівний
{$maxPageSize}.";
        return $alert;
    }

    return null;
}

function validate_pagination_after_query($currentPage, $pageSize, $collection)
{
    $alert = [
        'type' => 'error',
        'heading' => 'Пагінація',
    ];
    $count = $collection->count();

    if (($count === 0 && $currentPage !== 1)
        || ($count !== 0 && ($currentPage - 1) * $pageSize + 1 > $count)) {
        $alert['message'] = 'Номер сторінки вказаний неправильно.';
        return $alert;
    }

    return null;
}

function pagination(
    Request     $request,
    string|null $route,
    callable    $accessor,
    string      $name,
    string      $view,
    array       $additional
)
{
    $currentPage = intval($request['page'] ?? 1);
    $pageSize = intval($request['size'] ?? 10);

    $alert = validate_pagination_before_query($currentPage, $pageSize, 100);
    if ($alert) {
        if ($route) {
            return redirect()->route($route)->with([
                'alert' => $alert,
            ]);
        } else {
            return redirect_back()->with([
                'alert' => $alert,
            ]);
        }
    }

    $models = $accessor();
    $alert = validate_pagination_after_query($currentPage, $pageSize, $models);
    if ($alert) {
        if ($route) {
            return redirect()->route($route)->with([
                'alert' => $alert,
            ]);
        } else {
            return redirect_back()->with([
                'alert' => $alert,
            ]);
        }
```

```
    }

    return view($view, array_merge([
        $name => $models->skip(($currentPage - 1) * $pageSize)->take($pageSize),
        'pagination' => [
            'recordCount' => $models->count(),
            'pageSize' => $pageSize,
            'currentPage' => $currentPage,
            'spread' => 2,
        ]
    ], $additional));
}

function get_filters(Request $request)
{
    $filters = [];
    foreach ($request->all() as $parameterName => $parameterValue) {
        $type = mb_substr($parameterValue, 0, mb_strpos($parameterValue, '(') ?:
null);
        switch ($type) {
            case 'range':
                $configuration = explode('_',
get_string_between($parameterValue, '(', ')'));
                if (count($configuration) == 2) {
                    $filters[$parameterName] = [
                        'from' => $configuration[0],
                        'to' => $configuration[1],
                    ];
                }
                break;
            case 'list':
                $configuration = explode('_',
get_string_between($parameterValue, '(', ')'));
                if (count($configuration) > 0 && $configuration[0] !== '') {
                    $filters[$parameterName] = [
                        'values' => $configuration,
                    ];
                }
                break;
        }
    }
    return $filters;
}

function filter(Builder $builder, $filters)
{
    foreach ($filters as $filterName => $configuration) {
        switch ($configuration['type']) {
            case 'range':
                $builder->where(function ($query) use ($filterName,
$configuration) {
                    $query
                        ->where($filterName, '>=', $configuration['from'])
                        ->where($filterName, '<=', $configuration['to']);
                });
                break;
            case 'list':
                if (!$configuration['skip']) {
                    $builder->where(function ($query) use ($configuration) {
                        $query
                            ->whereIn($configuration['column'],
$configuration['values']);
                    });
```

```
                }
                break;
            }
        }
        display($builder->toSql());
        return $builder;
    }

    function asset_or_null($path)
    {
        if (File::exists($path)) {
            return asset($path);
        }
        return null;
    }

    function get_string_between($string, $start, $end)
    {
        $startIndex = mb_strpos($string, $start);
        $endIndex = mb_strrpos($string, $end);
        return mb_substr($string, $startIndex + 1, $endIndex - $startIndex - 1);
    }

    function redirect_back()
    {
        $referer = request()->header('referer');
        if (empty($referer)) {
            if (Auth::guest()) {
                return redirect()->home();
            } else {
                $user = Auth::user();
                if ($user->isAdmin()) {
                    return redirect()->route('admin.dashboard');
                } else {
                    return redirect()->home();
                }
            }
        } else {
            return redirect()->back();
        }
    }

    function get_settings(...$names)
    {
        $storedSettings = Setting::whereIn('name', [...$names])->get();
        $settings = [];
        foreach ($storedSettings as $storedSetting) {
            $settingName = $storedSetting->name;
            $settingValue = $storedSetting->value;

            if (array_key_exists($settingName, $settings)) {
                if (is_array($settings[$settingName])) {
                    $settings[$settingName][] = $settingValue;
                } else {
                    $settings[$settingName] = [$settings[$settingName],
    $settingValue];
                }
            } else {
                $settings[$settingName] = $settingValue;
            }
        }
        return $settings;
    }
```

```
function get_setting($name, $defaultValue = null)
{
    $settings = get_settings($name);
    $value = array_key_exists($name, $settings) ? $settings[$name] : null;
    return $value ?: $defaultValue;
}

function set_setting($name, $value)
{
    $settings = Setting::where('name', $name)->get();
    $isArray = is_array($value);
    if ($settings->count() === 1 &&
        ($isArray && count($value) === 1 || !$isArray)
    ) {
        $settings->first()->update([
            'value' => $isArray ? reset($value) : $value,
        ]);
        return;
    }

    if ($isArray) {
        foreach ($value as $arrayValue) {
            (new Setting([
                'name' => $name,
                'value' => $arrayValue,
            ]))->save();
        }
    } else {
        (new Setting([
            'name' => $name,
            'value' => $value,
        ]))->save();
    }
}

function failed_validation($validator, $bagName, $errorHeading)
{
    return redirect_back()
        ->withInput()
        ->withErrors($validator->getMessageBag(), $bagName)
        ->with([
            'alert' => [
                'type' => 'error',
                'heading' => $errorHeading,
                'message' => 'Дані не відповідають вимогам.',
            ],
        ]);
}

function upload_if_exists(Request $request, string $key, string $path)
{
    if ($request->hasFile($key) && $request->file($key)->isValid()) {
        $filename = $request->file($key)->hashName();
        $request->file($key)->storeAs("public/{$path}", $filename);
        return $filename;
    }
    return null;
}
```