МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему

«Односторінковий веб-додаток для інтернет-торгівлі друкованих видань»

Студент гр. ІНз-61с _____(підпис) Боруха Р.В.

Керівник практики _____(підпис, печатка бази практики) Проценко О.Б.

Керівник роботи _____(підпис) Проценко О.Б.

СУМИ 2020

Затверджую_____

Зав. кафедрою Довбиш А.С.

“_____” _____2020 г.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІНз-61с спеціальності “Інформатика” заочної форми навчання Борухи Руслана Віталійовича.

Тема: “Односторінковий веб-додаток для інтернет-торгівлі друкованих видань”

Затверджена наказом по СумДУ

№_____ от _____2020 г.

Зміст пояснювальної записки: вступ, постановка задачі, інформаційний огляд, вибір веб-технологій, інформаційна модель, проектування бази даних, програмна реалізація, висновок, список літератури.

Дата видачі завдання “_____”_____20 г.

Керівник випускної роботи _____ Проценко О.Б.

Завдання приняв до виконання _____ Боруха Р.В.

РЕФЕРАТ

Записка: 49 стор., 2 рис., 1 табл., 15 джерел.

Об’єкт дослідження — односторінковий веб-додаток.

Мета роботи — розробка односторінкового веб-додатку, так званого Single Page Application, що не перезавантажується при переході по посиланню, і працює подібно до настільного додатку.

Методи дослідження — емпіричний.

Результати — розроблено односторінковий веб-додаток для інтернет-торгівлі друкованих видань, що є схожим у використанні на настільний додаток. Дає змогу користувачам замовляти товари, а адміністратору додавати товари за допомогою окремої сторінки.

WEB-РЕСУРС, SINGLE PAGE APPLICATION, PHP, NGINX, POSTGRESQL, JAVASCRIPT

Зміст

ВСТУП

Інтернет-магазини користується попитом серед підприємців. Електронна торгівля значно скорочує витрати на політику збуту, оскільки товар пропонується споживачам без посередників. Окрім цього, Інтернет-магазин не потребує змінних витрат на оренду, комунальні послуги, зайвих трудових ресурсів, на відміну від точок роздрібу чи оптових точок. Товари майже не бувають на складі у надлишку чи в дефіциті, оскільки попит простежується оперативно завдяки переглядам конкретних товарів, цільовим діям на сайті, а можливість укладення договорів з фірмами про швидке постачання дає змогу скоротити площу складу.

Просування товару відбувається значно ефективніше, оскільки в мережі Інтернет чіткіший таргетинг і фокус зосереджується на конкретній цільовій аудиторії. Інтернет-магазин дозволяє вичерпно ознайомити потенційного споживача з усіма асортиментними одиницями, закликати до дії за допомогою кнопок, поштових розсилок, перетворювати його у постійного, лояльного клієнта, застосовуючи різноманітні інструменти комунікації.

Процес спілкування зі споживачем більш налагоджений, а зворотній зв’язок якісніший. Завдяки Інтернет-магазину можно орієнтуватися на групи споживачів з різними рівнями доходу, сегментувати ринок, а також застосовувати цінову диверсифікацію.

Беззаперечною перевагою Інтернет-магазину над точками роздрібу є його цілодобова доступність для споживача, а також скорочення часу на ознайомлення з асортиментом та процес купівлі. У клієнта відпадає необхідність отримувати консультацію від персоналу, адже кожний товар має вичерпну характеристику, опис та фото. Можливість резервування є також значною особливістю електронної торгівлі.

У даній роботі буде розглянуто процес створення інтернет-магазину, використовуючи технічний підхід односторінкового веб-додатку, який

полягає у взаємодії з користувачем за допомогою динамічного відображення контенту замість завантаження сторінки зі сервера. Цей підхід забезпечує уникнення переривання досвіду користування користувача між переходами на сторінки, значно збільшує швидкодію системи після завантаження і робить поведінку додатку більш схожою на настільну програму.

1.ПОСТАНОВКА ЗАДАЧІ

У даній роботі необхідно реалізувати односторінковий веб-додаток, який має наступні розділи:

- Головна сторінка, на якій відображаються 10 найпопулярніших за продажами товарів, список категорій товарів.
- Сторінка товару, що містить фото, назву, опис, вартість, кнопку, щоб додати в кошик
- Сторінка категорії зі списком товарів з пагінацією у цій категорії.
- Кошик з товарами, що містить список доданих товарів з можливістю вказати кількість і видалити.
- Кошик зберігається після перезавантаження сторінки.
- Пошук по товарам (відображається на окремій сторінці з пагінацією).
- Авторизація користувачів за допомогою Google.
- Оформлення замовлення товару тільки для авторизованих користувачів з полем для введення адреси доставки.
- Адміністратор має можливість додавання, видалення та редагування товарів.

2.ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Веб-додаток

Веб-додаток – це клієнт-серверна комп’ютерна програма, у якій клієнт (включаючий інтерфейс користувача і логіку клієнтської частини) виконується у веб-браузері.

До розповсюджених веб-додатків відносяться веб-пошта, онлайн торгівля, онлайн банкінг і онлайн аукціони.

Для того, щоб краще зрозуміти суттєву відмінність веб-додатків від веб-сайтів, наведемо визначення веб-сайту.

Веб-сайт – це колекція публічно доступних, зв’язаних сторінок, які мають одне ім’я домену. Веб-сайт можуть бути створені і підтримуватись однією особою, групою, бізнесом, організацією для задоволення різних цілей. Усі публічно доступні веб-сайти разом складають Всесвітню мережу (World Wide Web). Зазвичай інформація на веб-сайтах збирається таким чином, щоб користувачі могли легко орієнтуватися і отримувати знання, що відповідають їх потребам.

Класичними прикладами веб-сайтів є блоги, сайти новин, урядові та навчальні сайти.

Основними елементами веб-сайту є Hypertext Markup Language (далі HTML), Cascading Style Sheets (далі CSS) і JavaScript. Веб-сайти не потребують інших мов програмування чи баз даних.

На відміну від веб-сайту, веб-додатки більш чуйні до дій користувачів. Вони є інтерактивними, і забезпечують користувачам можливість маніпулювати даними і запрошувати різні дані[2].

Класичними прикладами веб-додатків є веб-пошта, сайти роздрібної торгівлі, текстові редактори, фото- та відео-редактори.

Веб-додатки охоплюють усі технології, які використовуються на веб-сайтах. Крім того вони використовують мови програмування такі, як PHP, NodeJS, Ruby, Python та бази даних.

2.2. Мережева архітектура «клієнт-сервер»

Архітектура “клієнт-сервер” – це розподілена структура додатків, яка розділяє завдання або робочі навантаження між постачальниками ресурсу чи послуги, що називаються серверами, і замовниками послуг, що називаються клієнтами. Часто клієнт і сервер комунікують через комп’ютерну мережу на окремому апаратному забезпеченні, проте клієнт і сервер можуть функціонувати на одній системі. Хост сервера запускає одну або декілька програм, ділячись своїми ресурсами з клієнтом. Клієнт запрошує контент чи послугу з сервера. Клієнт таким чином ініціює сесію з сервером, який чекає на вхідні запити. Прикладами додатків, реалізуючих клієнт-серверну архітектуру, є веб-пошта, мережевий друк і Всесвітня мережа.

Ролі “клієнт” і “сервер”

Характеристика “клієнт-сервер” описує взаємозв’язок між співпрацюючими програмами у додатку. Сервери класифікуються за сервісами, які вони забезпечують. Наприклад, веб-сервер обслуговує веб-сторінки, файловий сервер обслуговує файли на машині. Ресурсом спільного користування може бути будь-яке серверне програмне чи апаратне забезпечення. Обмін ресурсами складає сервіс.

Чи є машина клієнтом, сервером чи і тим і іншим, визначається характером програми, яка вимагає сервісних функцій. Наприклад, одна машина може в той самий час може виконувати веб-серверне та файл-серверне ПЗ для того, щоб обслуговувати різні дані клієнтів, які роблять різні запити. Клієнтське ПЗ також може комунікувати з сервером на одній машині.

Комунікація клієнта і сервера

Сервіс є абстракцією комп'ютерних ресурсів, і клієнт не повинен турбуватися про те, яким чином сервер виконує запит і відправляє відповідь. Клієнт повинен лише розуміти відповідь на основі протоколу програми, тобто вміст і форматування даних для запитуваної послуги.

Клієнт і сервер обмінюються даними за допомогою шаблону обміну повідомленнями "запит-відповідь". Клієнт відправляє запит, а сервер відправляє відповідь. Цей обмін повідомленнями є прикладом міжпроцесової взаємодії. Щоб комунікувати, машини повинні мати спільну мову і вони повинні слідувати правилам, за якими обидві сторони знають чого очікувати. Мова і правила комунікації визначені в комунікаційному протоколі. Усі клієнт-серверні протоколи оперують на прикладному рівні. Протоколи прикладного рівня визначають базові шаблони діалогу. Щоб ще більше формалізувати обмін даними, сервер може реалізовувати прикладний програмний інтерфейс (Application Programming Interface, далі API). API є шаром абстракції для доступу до сервісу. Обмеживши спілкуванням конкретним форматом, полегшується синтаксичний аналіз. З абстрагуванням доступу полегшується крос-платформний обмін даними.

Сервер може отримувати запити від багатьох різних клієнтів у короткий проміжок часу. Машина може виконувати обмежену кількість задач в будь-який момент і покладається на систему планування для визначення пріоритету вхідних запитів.

Існує Атака на відмову в обслуговуванні (Denial of service attacks, DoS-атака), яка полягає в експлуатації зобов'язання сервера обробляти запити для його перевантаження з надмірною кількістю запитів за одиницю часу. Для запобігання надмірного використання та максимізації доступності, серверне ПЗ може обмежувати доступність для клієнтів.

Переваги і недоліки

Перевагами даної архітектури є:

- Відсутність дублювання коду програми сервера програмами клієнту.
- Оскільки всі обчислення виконуються на сервері, вимоги до клієнтської машини знижуються.
- Усі дані зберігаються на сервері, який, як правило, захищений набагато краще більшості клієнтів. На сервері простіше організувати контроль повноважень, щоб дозволяти доступ до даних тільки клієнтам з відповідними правами доступу.

Недоліками даної архітектури є:

- Непрацездатність сервера може зробити непрацездатною всю обчислювальну мережу. Непрацездатним сервером слід вважати сервер, продуктивності якого не вистачає на обслуговування всіх клієнтів або сервер, що знаходиться на обслуговуванні, профілактиці і т. п.
- Підтримка роботи даної системи потребує окремого спеціаліста – системного адміністратора.
- Висока вартість обладнання.

Багаторівнева архітектура “клієнт-сервер”

Також існує багаторівнева архітектура “клієнт-сервер”, суть якої полягає в тому, що функція обробки даних винесена на декілька окремих серверів. Це дозволяє розділити функції зберігання, обробки і представлення даних для більш ефективного використання можливостей серверів і клієнтів і забезпечення вільного зв’язку.

Трирівнева архітектура

Окремим випадком багаторівневої архітектури є трирівнева архітектура програмного комплексу, яка передбачає наявності у ньому трьох компонентів: клієнта, сервера і сервера бази даних.

Даний вид архітектури складається з таких компонентів:

Клієнт (шар клієнта) – це інтерфейсний (зазвичай графічний) компонент комплексу, надається кінцевому користувачу. Цей рівень не повинен мати прямих зв'язків з базою даних (за вимогами безпеки і масштабованості), бути навантаженим основний бізнес-логікою (за вимогами масштабованості) і зберігати стан додатки (за вимогами надійності). На цей рівень зазвичай виноситься тільки найпростіша бізнес-логіка: інтерфейс авторизації, алгоритми шифрування, перевірка введених значень на допустимість і відповідність формату, нескладні операції з даними (сортування, угруповання, підрахунок значень) вже завантаженими на термінал.

Сервер додатків (середній шар) розташовується на другому рівні, на ньому зосереджена велика частина бізнес-логіки. Поза його залишаються тільки фрагменти, що експортуються на клієнта (термінали), а також елементи логіки, що знаходяться у базі даних (збережені процедури і тригери). Реалізація даного компонента забезпечується підпрограмним забезпеченням. Сервери додатків проектуються таким чином, щоб додавання до них додаткових примірників забезпечувало горизонтальне масштабування продуктивності програмного комплексу і не вимагає внесення змін до програмного коду програми.

Сервер баз даних (шар даних) забезпечує зберігання даних і виноситься на окремий рівень[13]. Реалізується, як правило, засобами систем керування базами даних (далі СКБД), підключення до цього компоненту забезпечується тільки з рівня сервера додатків.

У найпростіших конфігураціях усі компоненти або частина з них можуть бути суміщені на одному обчислювальному вузлі. У продуктивних конфігураціях як правило використовується виділений обчислювальний вузол для сервера баз даних або кластер серверів баз даних, для серверів

додатків – виділена група обчислювальних вузлів, до яких безпосередньо підключаються клієнти (термінали).

У порівнянні з дворівневою клієнт-серверної архітектурою або файл-серверною архітектурою трирівнева архітектура забезпечує, як правило, більшу масштабованість (за рахунок горизонтальної масштабованості сервера додатків і мультиплексування з'єднань), велику конфігурованість (за рахунок ізольованості рівнів один від одного). Реалізація програм, доступних з веб-браузера або з тонкого клієнта, як правило, має на увазі розгортання програмного комплексу в трирівневій архітектурі. При цьому зазвичай розробка трирівневих програмних комплексів складніше, ніж для дворівневих, також наявність додаткового підпрограмного забезпечення може накладати додаткові витрати в адмініструванні таких комплексів.

2.3. Односторінковий додаток

Односторінковий додаток (Single Page Application, далі SPA) – це веб-додаток, який взаємодіє з користувачем, динамічно переписуючи поточну сторінку, а не завантажуючи нові сторінки з сервера. Цей підхід дозволяє уникнути переривання роботи користувача між послідовними сторінками, змусивши програму поводитись більше як настільний додаток. У SPA або весь необхідний код витягується з завантаженням однієї сторінки, або відповідні ресурси динамічно завантажуються та додаються на сторінку за необхідності, як правило, у відповідь на дії користувача[14]. Сторінка не завантажується в будь-який момент процесу, а також не передає керування на іншу сторінку, хоча хеш розташування або API історії HTML5 можуть використовуватися для забезпечення сприйняття та навігації окремих логічних сторінок у програмі[12]. Взаємодія з односторінковою програмою часто включає динамічне спілкування з веб-сервером.

Технічний підхід

Основними елементами, що використовуються при побудові SPA, є:

- JavaScript фреймворки, такі як AngularJS, Ember.js, ExtJS, Knockout.js, Meteor.js, React та Vue.js.
- Роутінг. Навігація між представленнями відбувається на стороні клієнта. Для реалізації переходів по сторінках використовуються JavaScript бібліотеки.
- Шаблонізатор.
- HTML5.
- API для бекенда. Наприклад, в стилі REST.
- Asynchronous Javascript and XML (Ajax) для динамічного завантаження контенту.

Деякі SPA можуть бути виконані з локального файлу, використовуючи схему URI файлу. Це дає користувачам можливість завантажувати SPA з сервера та запускати файл з локального пристрою зберігання даних, незалежно від підключення сервера. Якщо такий SPA повинен зберігати та оновлювати дані, використовується Web Storage.

2.4. Підхід до архітектури мережевих протоколів REST

Передача репрезентативного стану (REST) – це архітектурний стиль програмного забезпечення, який визначає набір обмежень, які будуть використані для створення веб-служб. Веб-сервіси, що відповідають архітектурному стилю REST, називаються послугами RESTful Web, забезпечують сумісність між комп'ютерними системами в Інтернеті. Веб-сервіси RESTful дозволяють замовникам отримувати доступ та маніпулювати текстовими поданнями веб-ресурсів за допомогою рівномірного та заздалегідь визначеного набору операцій без стану. Інші види веб-служб, такі як веб-сервіси SOAP, піддають власні довільні набори операцій.

"Веб-ресурси" вперше були визначені у Всесвітній павутині як документи або файли, визначені за їх URL-адресами. Однак сьогодні вони мають набагато більш загальне та абстрактне визначення, яке охоплює будь-яку річ чи сутність, які можна будь-яким чином ідентифікувати, назвати, звертатись чи обробляти в Інтернеті. У веб-службі RESTful запити, що надсилаються до URI ресурсу, будуть вимагати відповідь з корисним навантаженням, відформатованим у HTML, XML, JSON або інший формат. Відповідь може підтвердити, що деякі зміни були внесені до збереженого ресурсу, і відповідь може забезпечити гіпертекстові посилання на інші пов'язані ресурси або набори ресурсів. Коли використовується HTTP, а це найбільш часто, доступними операціями (методами HTTP) є GET, HEAD, POST, PUT, PATCH, DELETE, CONNECT, OPTIONS та TRACE.

Використовуючи протокол без стану та стандартні операції, системи RESTful мають на меті швидку продуктивність, надійність та можливість зростати за рахунок повторного використання компонентів, якими можна керувати та оновлювати, не впливаючи на систему в цілому, навіть під час її роботи.

Термін “передача стану репрезентації” був введений і визначений у 2000 році Роєм Філдінгом у своїй докторській дисертації. Дисертація

Філдінга пояснила принципи REST, які були відомі як "HTTP-об'єктна модель", починаючи з 1994 року, і використовувались при розробці стандартів HTTP 1.1 та Уніфікованих ідентифікаторів ресурсів (URI). Термін призначений для викликати образ поведінки добре розробленої веб-програми: це мережа веб-ресурсів, де користувач просувається через додаток, вибираючи ідентифікатори ресурсів, такі як http://www.example.com/articles/21 та такі операції з ресурсами, як GET або POST (переходи стану додатків), в результаті чого відображення наступного ресурсу (наступний стан програми) передається кінцевому користувачеві для його використання.

Архітектурні властивості

Обмеження архітектурного стилю REST впливають на такі архітектурні властивості:

- Продуктивність у взаємодії компонентів, що може бути домінуючим фактором у сприйнятті користувачем продуктивності та ефективності мережі.
- Масштабованість, що дозволяє підтримувати велику кількість компонентів та взаємодії між компонентами.
- Простота однорідного інтерфейсу.
- Модифікованість компонентів для задоволення мінливих потреб (навіть під час роботи програми).
- Видимість зв'язку між компонентами за допомогою сервісних агентів.
- Портативність компонентів шляхом переміщення програмного коду з даними.
- Надійність в стійкості до відмов на системному рівні за наявності відмов всередині компонентів, з’єднувачів або даних.

Архітектурні обмеження

Шість керівних обмежень визначають систему RESTful. Ці обмеження обмежують способи, якими сервер може обробляти та відповідати на запити клієнта, завдяки чому, діючи в рамках цих обмежень, система набуває бажаних нефункціональних вимог, таких як продуктивність, масштабованість, простота, модифікованість, видимість, портативність та надійність. Якщо система порушує будь-який з необхідних обмежень, її не можна вважати RESTful.

Клієнт-серверна архітектура

Принцип, що стоїть перед обмеженнями клієнт-сервер, – це розділення відповідальності. Відокремлення відношень з користувальницьким інтерфейсом від відношень зберігання даних покращує портативність користувацьких інтерфейсів на безлічі платформ. Це також покращує масштабованість, спрощуючи серверні компоненти. Але, мабуть, найбільш важливим для мережі Інтернет є те, що розділення дозволяє компонентам розвиватися незалежно, підтримуючи тим самим потребу в Інтернет-масштабі кількох організаційних областей.

Відсутність стану

Зв'язок клієнт-сервер обмежений тим, що на сервері між запитами не зберігається контекст клієнта. Кожен запит будь-якого клієнта містить всю інформацію, необхідну для обслуговування запиту, і стан сеансу зберігається у клієнта. Стан сеансу може бути перенесений сервером на іншу службу, таку як база даних, щоб підтримувати стійкий стан протягом періоду і дозволяти аутентифікацію. Клієнт починає надсилати запити, коли готовий здійснити перехід до нового стану. Поки один або більше запитів є невиконаними, клієнт вважається перехідним. Представлення кожного стану програми містить посилання, які можуть бути використані наступного разу, коли клієнт вирішить ініціювати новий стан переходу.

Кешованість

Як і у Всесвітній мережі, клієнти та посередники можуть кешувати відповіді. Тому відповіді повинні неявно або явно визначати себе як кешовані або не кешовані, щоб запобігти отримання клієнтами застарілих або невідповідних даних у відповідь на подальші запити. Добре кероване кешування частково або повністю виключає деякі клієнт-серверні взаємодії, ще більше покращуючи масштабованість та продуктивність.

Багатошарова система

Клієнт не може сказати, підключений він безпосередньо до кінцевого сервера чи до посередника вздовж шляху. Це означає, що клієнт не знає, чи спілкується він з проміжним або фактичним сервером. Отже, якщо проксі-сервер або сервіс балансування навантаження розміщений між клієнтом і сервером, це не вплине на їх зв’язок і не буде необхідності оновлення коду клієнта або сервера. Посередницькі сервери можуть покращити масштабованість системи, ввімкнувши балансування навантаження та надання спільних кешів. Також можна додати безпеку як шар поверх веб-служб, а потім чітко відокремити бізнес-логіку від логіки безпеки. Додаючи безпеку як окремий рівень, застосовується політика безпеки. Нарешті, це також означає, що сервер може викликати кілька інших серверів, щоб генерувати відповідь клієнту.

Код на вимогу (необов’язково)

Сервери можуть тимчасово розширити або налаштувати функціональність клієнта шляхом передачі виконуваного коду: наприклад, складених компонентів, таких як аплети Java, або сценарії на стороні клієнта, такі як JavaScript.

Однорідний інтерфейс

Обмеження однорідності інтерфейсу є основним для проектування будь-якої системи RESTful. Це спрощує та відокремлює архітектуру, що дозволяє кожній частині розвиватися незалежно. Чотири обмеження для цього єдиного інтерфейсу:

Ідентифікація ресурсу в запитах

Окремі ресурси ідентифікуються у запитах, наприклад, використовуючи URI в веб-службах RESTful. Самі ресурси концептуально відокремлені від представлених даних, які повертаються клієнту. Наприклад, сервер може надсилати дані зі своєї бази даних у вигляді HTML, XML або як JSON – жодне з яких не є внутрішнім представництвом сервера.

Маніпулювання ресурсами через представлення

Якщо клієнт має представлення ресурсу, включаючи будь-які додані метадані, він має достатньо інформації для зміни або видалення ресурсу.

Самоописові повідомлення

Кожне повідомлення містить достатньо інформації, щоб описати, як обробити повідомлення. Наприклад, який парсер треба використовувати може бути визначено за допомогою MIME типу.

Гіпермедіа як двигун стану додатку

Отримавши доступ до початкового URI для REST додатку – аналогічного веб-користувачу, який здійснює доступ до домашньої сторінки веб-сайту – клієнт REST повинен мати можливість динамічно використовувати надані сервером посилання, щоб виявити всі наявні дії та ресурси, необхідні йому. По мірі доступу сервер відповідає текстом, який включає гіперпосилання на інші дії, які зараз доступні. Немає необхідності, щоб клієнт був жорстко кодований інформацією щодо структури та динаміки програми.

Застосування з веб-службами

API веб-служб, які дотримуються архітектурних обмежень REST, називаються API RESTful. RESTful API на основі HTTP визначаються такими аспектами:

- Базовий URI, такий як http://api.example.com/collection/.
- Стандартні методи HTTP (наприклад, GET, POST, PUT, PATCH і DELETE).
- MIME тип, який визначає елементи даних про перехід стану (наприклад, Atom, мікроформати, application/vnd.collection + json). Поточне представлення повідомляє клієнту, як складати запити на переходи до всіх наступних доступних станів додатків. Це може бути простим, як URI, або складним, як аплет Java.

Зв’язок між URI та HTTP-методами

Наступна таблиця показує, як методи HTTP зазвичай використовуються в API RESTful:

HTTP-метод	Ресурс колекції, наприклад https://api.example.com/collection/	Ресурс-член, наприклад https://api.example.com/collection/item3
GET	Отримання колекції ресурсів у тілі відповіді	Отримання ресурсу-члена у тілі відповіді
POST	Створення ресурсу-члена у ресурсі колекції, використовуючи інструкції в тілі запиту. URI створеного ресурсу-члена автоматично присвоюється та повертається у поле заголовка відповіді Location.	Створення ресурсу-члена у ресурсі колекції, використовуючи інструкції в тілі запиту. URI створеного ресурсу-члена автоматично присвоюється та повертається у поле заголовка відповіді Location.

PUT	Заміна усіх представлень ресурсів-членів ресурсу колекції представленням у тілі запиту або створення ресурсу колекції, якщо його не існує.	Заміна усіх представлень ресурсу-члена або створення ресурсу-члена представленням у тілі запиту, якщо його не існує.
PATCH	Оновлення усіх представлень ресурсів-членів колекційного ресурсу, використовуючи інструкції у тілі запиту або створення колекційного ресурсу, якщо його не існує.	Оновлення усіх представлень ресурсу-члену або створення ресурсу-учасник, якщо його не існує, скориставшись інструкціями в тілі запиту.
DELETE	Видалення всіх представлень ресурсів-учасників ресурсу колекції.	Видалення всіх представлень ресурсу-учасника.

Таблиця 1. Зв’язок між URI та HTTP-методами

Метод GET є безпечним, тобто застосування його до ресурсу не призводить до зміни стану ресурсу.

Методи GET, PUT і DELETE є ідемпотентними, тобто застосувавши їх кілька разів до ресурсу, це призведе до такої ж зміни стану ресурсу, як і один раз їх застосування, хоча відповідь може відрізнятися.

На відміну від веб-сервісів на основі SOAP, немає "офіційного" стандарту для RESTful Web API. Це тому, що REST – це лише архітектурний стиль, а SOAP – протокол. REST сам по собі не є стандартом, але в реалізації RESTful використовуються стандарти, такі як HTTP, URI, JSON та XML. Багато розробників також описують свої API як RESTful, навіть якщо ці API фактично не відповідають усім архітектурним обмеженням, описаним вище (особливо єдиному обмеженню інтерфейсу).

2.5. Технічний підхід Ajax

AJAX – технічний підхід, що розшифровується як Asynchronous JavaScript and XML, і описує набір методів розробки, що використовуються для створення веб-сайтів та веб-додатків. Основна функція AJAX полягає в асинхронному оновленні веб-контенту, тобто веб-браузеру користувача не потрібно перезавантажувати всю веб-сторінку, коли потрібно змінити лише невелику частину вмісту на сторінці[6].

Одним з найбільш повсюдних прикладів асинхронного оновлення є функція Google «Запропонувати Google». При ввведенні пошукового запиту у рядок пошуку Google веб-сайт Google автоматично починає пропонувати параметри автоматичного заповнення під час введення, тобто AJAX у дії. Вміст сторінки змінюється (у цьому випадку параметри автоматичного заповнення на панелі пошуку), не оновлюючи її вручну (те, що зробить Google Suggest недоцільним у використанні). Такі функції, як Google Suggest, є основоположною частиною сучасного веб-перегляду, що вказує на те, наскільки важливим є AJAX у веб-розробці. На додаток до Google Suggest, AJAX зазвичай використовується для оновлення таких функцій, як панелі стану та повідомлень, онлайн-форми, розділи коментарів та опитування.

3.ВИБІР ВЕБ-ТЕХНОЛОГІЙ

3.1. PHP-фреймворк Laravel

Laravel – це вільний PHP-фреймворк з відкритим вихідним кодом, призначений для розробки веб-додатків за допомогою архітектурного шаблону проектування модель-вигляд-контролер (MVC) [9].

Цей шаблон полягає у розділенні даних додатку, інтерфейсу та логіки управління. Модель використовується для взаємодії з базою даних, а база даних може бути як SQL, NoSQl, файл JSON або інший ресурс. Контролер містить логіку, наприклад, як перевірити дані форми та зберегти ресурс до бази даних, взаємодіючи через Model. Перегляд – це інтерфейс користувача (UI) програми, що містить HTML або розмітку вигляду. Він також може мати логіку, наприклад, цикли та умовні оператори. Двигуни шаблонів використовуються для вбудовування логіки у Views. У Laravel є двигун шаблонів Blade, який використовується для додавання логіки всередині вигляд.

Laravel є модульною системою з виділеним менеджером залежностей, з різними способами доступу до реляційних баз даних, утиліти, що допомагають у розгортанні та обслуговуванні додатків, та його орієнтація на синтаксичний цукор. Він надає безліч функцій, таких як повна система аутентифікації, міграція баз даних, потужна ORM, пагинація. Вихідний код Laravel розміщується на GitHub та ліцензується на умовах ліцензії MIT[1].

Маршрути

У laravel є два файли маршрутів web.php та api.php. Файл web.php використовується для реєстрації всіх веб-маршрутів, таких як mywebsite.com/about або mywebsite.com/contact. api.php використовується для реєстрації всіх маршрутів, пов'язаних з api.

Контролери та моделі

Моделі зберігаються в каталозі додатків, і існує модель користувача за замовчуванням, яка постачається з laravel для цілей аутентифікації. app\Http\Controllers має всі контролери та є деякі контролери за замовчуванням для аутентифікації.

Вигляди

Вигляди можна знайти в ресурсах / виглядах і welcome.blade.php - це стандартний вид, який надає laravel. Вигляди Laravel мають розширення .blade.php.

Service Container

Сервісні контейнери це інструмент для управління залежностями та реалізації впровадження залежностей (dependency injection), що означає, що певний клас впроваджується в інший за допомогою конструктора класу або сетера. Сервісний контейнер це клас, де реєструються залежності, які будуть в подальшому «ін“єкціюватися» у інші класи. Цю функціональність реалізовано за допомогаю Reflection API, який дає змогу проводити повну інтроспекцію класів, інтерфейсів, функцій, методовів, модулів[10].

Міграції

Для того, щоб вам не доводилося вручну створювати таблиці або додавати окремі колонки до таблиць, створено функцію виконання міграцій бази даних. Ця функція надає змогу відтворити описану у відповідних файлах схему бази даних.

Посів тестових даних (Seeding)

Для наповнення бази даних тестовими даними використовується метод Seeding. Класи, що наслідують базовий клас Seeder, мають метод run(), що відповідає за вставку даних до таблиць бази даних. Вставка даних може бути реалізована за допомогою Query Builder, Eloquent (ORM) або за допомогою

Фабрик моделей, що є більш зручним методом, оскільки дає змогу генерувати велику кількість даних.

3.2. JavaScript бібліотека React

React – це бібліотека JavaScript для побудови інтерфейсів користувача. Він підтримується Facebook та спільнотою окремих розробників та компаній. React може бути використаний в якості основи при розробці односторінкових або мобільних додатків, оскільки він оптимальний для отримання швидко змінюваних даних, які потрібно записати. Однак отримання даних – це лише початок того, що відбувається на веб-сторінці, тому складні програми React зазвичай вимагають використання додаткових бібліотек для управління станом, маршрутизації та взаємодії з API[4].

Компоненти

Код реагування складається з об'єктів, які називаються компонентами. Компоненти можуть бути передані певному елементу в DOM за допомогою бібліотеки React DOM. Під час візуалізації компонента можна передати значення, відомі як "props". Э два способи декларування компонентів у React: через функціональні компоненти або компоненти на основі класу. Функціональні компоненти оголошуються функцією, яка потім повертає JSX. Компоненти на основі класу оголошуються за допомогою класів ES6. Вони також відомі як "stateful" компоненти, оскільки їх стан може містити значення у всьому компоненті і може бути переданий дочірнім компонентам через “props”.

Віртуальний DOM

Ще одна примітна особливість – використання віртуальної моделі об'єкта документа або віртуальної DOM. React створює кеш даних структури пам’яті, обчислює отримані відмінності, а потім ефективно оновлює відображений DOM браузера. Це дозволяє програмісту записувати код так, ніби вся сторінка відображається при кожній зміні, в той час як бібліотека React візуалізує лише підкомпоненти, які змінюються.

Методи життєвого циклу

Методи життєвого циклу - це хуки (hooks), що дозволяють виконувати код у встановлені моменти протягом життя компонента.

- shouldComponentUpdate дозволяє розробнику запобігати непотрібному повторному рендерингу компонента шляхом повернення false, якщо візуалізація не потрібна.
- componentDidMount визивається, коли компонент "змонтований" (компонент був створений в інтерфейсі користувача, часто, пов'язуючи його з вузлом DOM). Це зазвичай використовується для запуску завантаження даних з віддаленого джерела через API.
- componentWillUnmount викликається безпосередньо перед тим, як компонент буде зірваний або "відключений". Це зазвичай використовується для очищення залежностей від компонента, які не будуть самі собою усунені при размонтуванні компонента (наприклад, видалення будь-яких екземплярів “setInterval”, які пов'язані з компонентом).
- render – найважливіший метод життєвого циклу і єдиний необхідний у будь-якому компоненті. Зазвичай його визивають щоразу, коли стан компонента оновлюється, що відображається в інтерфейсі користувача.

JSX

JSX або JavaScript XML – це розширення до синтаксису мови JavaScript. За зовнішнім виглядом, схожий на HTML, JSX пропонує спосіб структурування візуалізації компонентів за допомогою синтаксису, знайомого багатьом розробникам. Компоненти React, як правило, записуються за допомогою JSX, хоча це не є обов’язковим (компоненти також можуть бути записані у чистому JavaScript). JSX схожий на інший синтаксис розширення, створений Facebook для PHP під назвою XHP.

React Hooks

Хуки – це функції, які дозволяють розробникам змінювати стан і викликати методи життєвого циклу у функціональних компонентах. Хуки не працюють у класах – вони дозволяють вам використовувати React без класів. React забезпечує вбудовані хуки типу useState. Ви також можете створити власні хуки для повторного використання поведінки з наявністю стану між різними компонентами[5].

3.3. Веб-сервер Nginx

Nginx – веб-сервер, який також може використовуватися як зворотний проксі, система балансування навантаження, поштовий проксі та HTTP кеш. Значна частина веб-серверів використовує Nginx, часто як систему балансування навантаження.

Nginx може бути розгорнутий для обслуговування динамічного контенту HTTP в мережі за допомогою FastCGI, обробників SCGI для скриптів, серверів додатків WSGI або модулів Phusion Passenger, і він може слугувати в якості системи балансування завантаження програмного забезпечення[15].

Nginx використовує асинхронний підхід, керований подіями, а не потоками для обробки запитів. Модульна архітектура, орієнтована на події Nginx, може забезпечити більш передбачувані показники роботи при великих навантаженнях.

Особливості

- Можливість обробляти більше 10 000 одночасних з'єднань із низьким розміром пам’яті (~ 2,5 Мб на 10 000 неактивних HTTP-з'єднань).
- Обробка статичних файлів, файлів індексів та автоматична індексація.
- Зворотний проксі з кешуванням.
- Балансування навантаження за допомогою внутрішніх перевірок.
- TLS / SSL з підтримкою зшивання SNI та OCSP через OpenSSL.
- Підтримка FastCGI, SCGI, uWSGI з кешуванням.
- Підтримка gRPC з березня 2018 року, версія 1.13.10.
- Віртуальні на основі імен та IP-адрес сервери.
- Сумісний з IPv6.

- WebSockets починаючи з 1.3.13, включаючи функцію зворотного проксі-сервера та балансування навантаження програм WebSocket.
- Оновлення HTTP/1.1 (101 протокол комутації), підтримка протоколу HTTP/2.
- Переписування та переадресація URL-адрес.

3.4. Система керування базою даних PostgreSQL

PostgreSQL, також відомий як Postgres, - це вільна та відкрита система управління реляційними базами даних, що підкреслює розширюваність та дотримання технічних стандартів.

Він призначений для роботи з різними навантаженнями, від окремих машин до сховищ даних або веб-сервісів з багатьма одночасними користувачами[11]. Це база даних за замовчуванням для сервера macOS, а також доступна для Linux, FreeBSD, OpenBSD та Windows.

PostgreSQL пропонує транзакції з властивостями Atomicity (Атомарність), Consistentcy (Узгодженість), Isolation (Ізольованість), Durability (Довговічність), автоматично оновлювані представлення даних, матеріалізовані представлення, тригери, зовнішні ключі та збережені процедури.

Psql

Основним фронтовим для PostgreSQL є програма командного рядка psql, яка може використовуватися для введення запитів SQL безпосередньо або виконання їх з файлу. Крім того, psql надає ряд метакоманд та різних функцій, подібних до оболонок, для полегшення написання сценаріїв та автоматизації широкого спектру завдань; наприклад, завершення вкладки імен об'єктів та синтаксису SQL.

PgAdmin

Пакет pgAdmin – це безкоштовний графічний користувальницький інтерфейс (GUI) для адміністрування PostgreSQL, який підтримується на багатьох комп'ютерних платформах.

4.ІНФОРМАЦІЙНА МОДЕЛЬ

Для представлення взаємодії користувачів з веб-додатком, розроблено його інформаційну модель – UML-діаграму варіантів використання інтернет-магазину (рис 4.1).

На діаграмі відображено, що система може мати користувачів трьох ролей: незареєстрованого користувача, зареєстрованого користувача, адміністратора. Для незареєстрованого користувача недоступно оформлення замовлення. Користувач ролі Адміністратор має привілеї над усіма іншими користувачами, а саме редагування контенту.

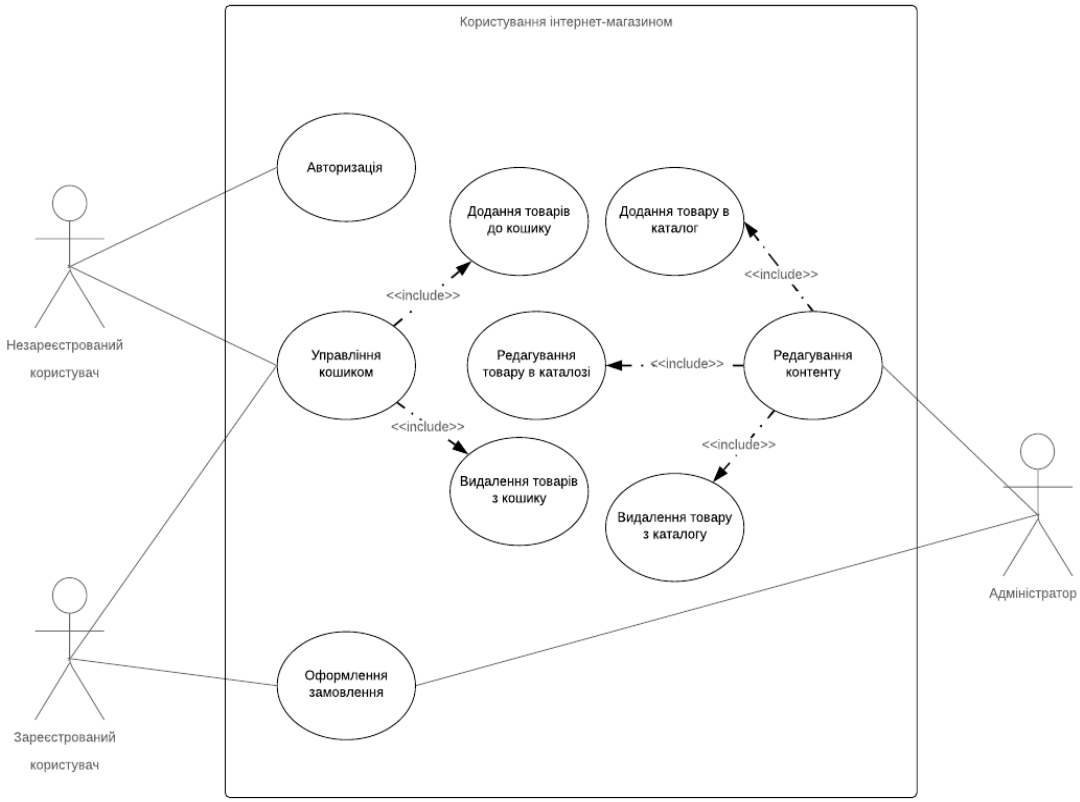

Рис 4.1. UML-діаграма варіантів використання

5.ПРОЕКТУВАННЯ БАЗИ ДАНИХ

База даних – це організована колекція даних, яка зберігається та доступна в електронному вигляді з комп’ютерної системи.

Науковці можуть класифікувати системи управління базами даних відповідно до моделей баз даних, які вони підтримують. Реляційні бази даних стали домінуючими у 1980-х роках.

Дані реляційної моделі зберігаються у вигляді рядків та стовпців у таблицях, рядки яких ідентифікуються унікальним ключем. Переважна більшість використовує SQL для запису та запиту даних.

Програмною системою, що використовується для підтримки баз даних, є система керування базами даних (СКБД). У багатьох реляційних системах баз даних є можливість використання SQL (Структурована мова запитів) для запитів і підтримки бази даних.

База даних інформаційної системи має такі сутності:

- Користувач.
- Замовлення.
- Продукт.
- Категорія.
- Файл.

Для відображення зв’язку між сутностями було побудовано ER-діаграму (рис 5.1). На ній відображено сутності та зв’язки між цими сутностями [7].

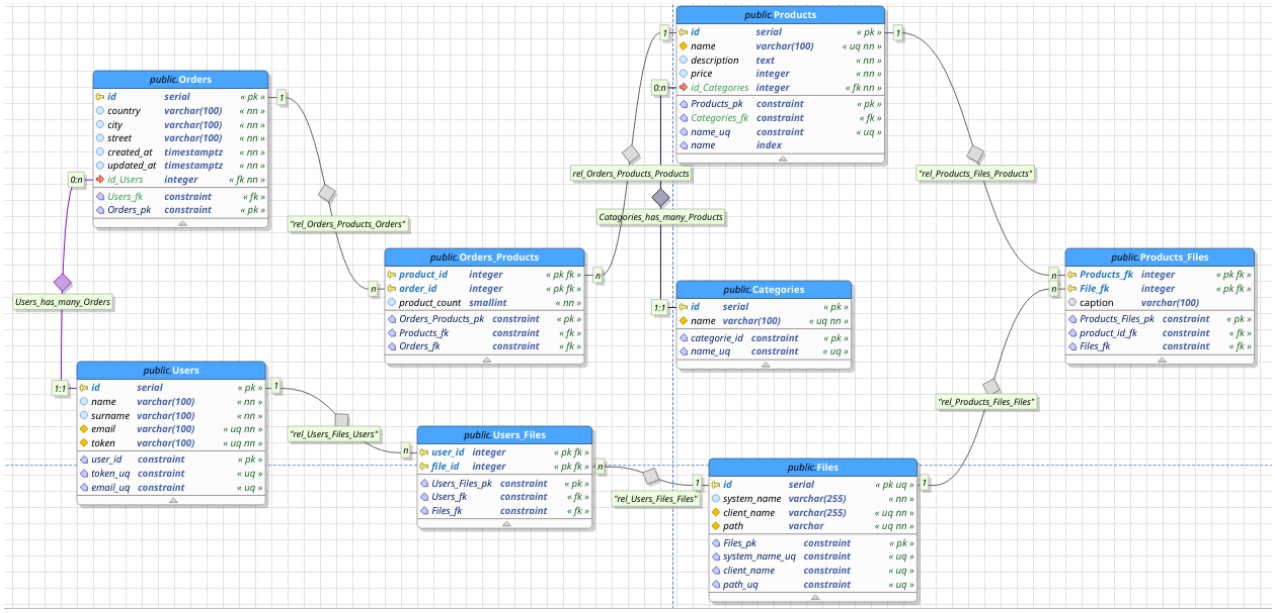

Рис 5.1. ER-діаграма бази даних веб-додатку

6. ПРОГРАМНА РЕАЛІЗАЦІЯ

6.1. Серверна частина

Для розробки серверної частини буде використовуватися фреймворк Laravel, який вимагає встановлення PHP щонайменше 7.2.5 версії, консольну утиліту для встановлення PHP-пакетів Composer і деякі PHP-розширення, про які можна дізнатися в документації фреймворку.

Встановимо Laravel проект за допомогою Composer[8]:

composer create-project --prefer-dist laravel/laravel blog

Імплементуємо міграції бази даних:

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

Код вище повторюється у всіх наступних класах міграцій.

Таблиця користувачів:

```
class CreateUsersTable extends Migration
{
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name', 100);
            $table->string('surname', 100)->nullable();
            $table->string('email', 100)->unique();
            $table->string('token', 100)->unique();
        });
    }


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

Таблиця замовлень:

```
class CreateOrdersTable extends Migration
{
    public function up()
    {
        Schema::create('orders', function (Blueprint $table) {
            $table->increments('id');
            $table->string('country', 100);
```

```
            $table->string('city', 100);
            $table->string('street', 100);
            $table->integer('user_id');
            $table->foreign('user_id')->references('id')->on('users');
            $table->timestamps();
        });
    }

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

Та інші за зразком.

Імплементуємо фабрики (factories), які будуть використовуватися для наповнення бази даних тестовими даними:

```
<?php

use App\Category;
use Faker\Generator as Faker;

Код вище повторюється у всіх наступних фабриках.

Фабрика категорій:

$factory->define(Category::class, function (Faker $faker) {
    return [
        'name' => $faker->unique()->word()
    ];
});
```

Фабрика файлів:

```
$factory->define(File::class, function (Faker $faker) {
    $system_name = $faker->unique->word(5, 15);

    return [
        'system_name' => $system_name,
        'path' => "https://picsum.photos/seed/$system_name/1200/675"
    ];
});
```

Та інші фабрики за зразком.

Імплементуємо сіди (seeds) для наповнення бази даних:

Seeder замовлень:

```
<?php

use Illuminate\Database\Seeder;

class OrderTableSeeder extends Seeder
{
    protected $users;
    protected $products_max_id;
```

```
        function __construct()
        {
            $this->users = App\User::all();
            $this->products_max_id = $this->users->count();;
        }

        protected function getRandomIdArray() {
    // певна імплементація, що не є важливою
        }

        public function run()
        {
            foreach ($this->users as $user) {
                $order = factory(App\Order::class, rand(0, 3))->make();
                $user->orders()->saveMany($order);
            }

            $orders = App\Order::all();

            foreach ($orders as $order) {
                $products_id_arr = $this->getRandomIdArray();

                foreach ($products_id_arr as $product_id) {
                    $order->products()->attach($product_id, ['count' =>
rand(1, 10)]);
                }
            }
        }
    }
```

DatabaseSeeder (клас, що запускає всі інші класи):

```
class DatabaseSeeder extends Seeder
{
    public function run()
    {
        $this->call(UserTableSeeder::class);
        $this->call(CategoryTableSeeder::class);
        $this->call(ProductTableSeeder::class);
        $this->call(OrderTableSeeder::class);
    }
}
```

Імплементуємо моделі, які будуть виконувати функцію шаблону проектування “Active Record”. Кожен клас моделі має відповідну таблицю у базі даних, що надалі дає змогу створювати нові рядки в таблиці за допомогою відповідного методу.

Модель користувача:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;
class User extends Model
{
```

```
        public function files()
        {
            return $this->belongsToMany('App\File')->using('App\UserFile');
        }

        public function orders()
        {
            return $this->hasMany('App\Order');
        }


        public $timestamps = false;
    }
```

Усі інші моделі реалізовано за таким же принципом.

Імплементуємо контролери, що відповідають за обробку запитів. Один контролер повинен обробляти операції створення, читання, оновлення, видалення лише однією сутності.

Контролер категорій:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\DB;
use App\Category;
use Config;

class CategoryController extends Controller
{
    public function index()
    {
        return Category::all();
    }

    public function products(Request $request, $id) {
        $category = Category::findOrFail($id);
        $perPageParam = $request->input('per_page');
        $maxCountPerPage = Config::get('constants.maxItemCountPerPage');
        $perPage = isset($perPageParam) && ($perPageParam <=
$maxCountPerPage) ?
            $perPageParam :
            Config::get('constants.itemCountPerPage');

        return $category
            ->products()
            ->select('products.id', 'products.name', 'products.price',
'files.path as img')
            ->join('file_product', 'products.id',
'file_product.product_id')
            ->join('files', 'files.id', 'file_product.file_id')
            ->paginate($perPage);
    }
}
```

Усі інші контролери реалізовано за таким же принципом.

Імплементуємо Middleware, що відповідає за обробку запиту перед тим, як він буде оброблений за допомогою відповідного методу контролеру, що є дуже зручним для того, щоб дописати дані до об’єкту запиту або повернути певну відповідь без подальшої обробки методом контролера.

Кастомна аутентифікація:

```
<?php

namespace App\Http\Middleware;

use Closure;
use App\User;

class CustomAuth
{
    public function handle($request, Closure $next)
    {
        $token = $request->header('Authorization');
        $user = User::where('token', $token);

        if (User::where('token', $token)->exists()) {
            $request->merge(['userId' => $user->value('id')]);

            return $next($request);
        } else {
            return response('Invalid token', 401);
        }
    }
}
```

Імплементуємо роутинг, що відповідає за те, яким чином певний URL буде оброблятися. Логіка обробки запитів поміщена у відповідні контролери. Хоча може бути реалізована колбек-функцією, що не є найкращою практикою. Оскільки робить код менш структурованим та позбавляє можливості використання логіки обробки запитів для інших URL, що може використовуватися у контролері за допомогою виклику інших методів.

```
use Illuminate\Http\Request;

Route::middleware(['api'])->group(function () {
    Route::prefix('categories')->group(function () {
        Route::get('/{id}/products', 'CategoryController@products');
        Route::get('/', 'CategoryController@index');
    });

    Route::prefix('products')->group(function () {
        Route::get('/top/{count?}', 'ProductController@top');
        Route::get('/{id}', 'ProductController@show');
    });
```

```
          Route::prefix('profile')->middleware(['custom_auth'])-
>group(function () {
              Route::get('/', 'ProfileController@index');
              Route::post('/', 'ProfileController@store');
          });

          Route::post('/signin', 'SigninController@store');
          Route::post('/signup', 'SignupController@store');
      });
```

6.2. Клієнтська частина

Для початку встановимо Node.js, середовище виконання JavaScript, яке виконує JavaScript на стороні серверу, і npm, пакетний менеджер Node.js, який використовується за замовчуванням.

Для ініціалізації проекту на фреймворку React, буде використовуватися найбільш розповсюджений шаблон Create-react-app. Він налаштовує середовище розробки, використовуючи Webpack, Babel, ESLint. Тим самим значно спрощуючи, прискорюючи процес ініціалізації проекту та даючи змогу фокусування тільки на розробці.

Розглянемо найбільш вагомі пакети, що використовує цей шаблон.

Webpack – пакет, що збирає модулі проекту та генерує граф залежностей, дозволяючи використовувати модульний підхід у веб-розробці. Він перетворює модулі у статичні зовнішні ресурси, також трансформуючи зовнішні ресурси, такі як HTML, CSS, зображення, якщо включені відповідні завантажувачі.

Babel – пакет, що відповідає за JavaScript транспайлінг – процес, що перетворює код однієї мови на код подібної, таким чином забезпечуючи підтримку нових специфікацій мови програмування.

ESLint – пакет, що відповідає за аналіз статичного коду з ціллю виявлення проблемних патернів, знайдених у коді.

Встановимо початковий проект Create-react-app[3]:

```
npx create-react-app my-app
```

Імплементуємо модуль контексту, який відповідає за те, щоб певний компонент міг мати доступ до пропсів без передавання пропсів за допомогою проміжних рівнів:

AppContext:

```
import React from 'react';

const initContext = {};

const AppContext = React.createContext(initContext);
```

```
export default AppContext;
```

Імплементуємо модуль утиліт:

Хуки, що допомагають облегчити використання класів в якості компонентів, оскільки дають змогу розбити компонент на малі функції по призначенню :

useStateWithLocalStorage:

```
import { useState, useEffect } from 'react';

const useStateWithLocalStorage = localStorageKey => {
    const [value, setValue] = useState(
        JSON.parse(localStorage.getItem(localStorageKey)) || {}
    );

    useEffect(() => {
        localStorage.setItem(localStorageKey, JSON.stringify(value));
    }, [value]);

    return [value, setValue];
};

export default useStateWithLocalStorage;
```

Точка входу index.js:

```
import React, { useEffect, useReducer } from 'react';
import ReactDOM from 'react-dom';
/* велика кількість імпортів компонентів, функцій, стилів, що не є
важливим для демонстрації принципу реалізації
*/
const { PROFILE_ROUTE } = routes;
const SET_LOGIN = 'SET_LOGIN';
const SET_GUEST = 'SET_GUEST';
const SET_LOADING = 'SET_LOADING';
const authInitState = {};

function App() {
    const [bag, setBag] = useStateWithLocalStorage('saveProducts');
    let appContext = { bag, setBag };
    const [authState, dispatch] = useReducer(authReducer,
authInitState);
    const token = localStorage.getItem(AUTH_TOKEN);

    const handleUserLogin = userData => {
        dispatch({ type: SET_LOGIN, payload: userData });
    };

    const handleUserLogout = () => {
        dispatch({ type: SET_GUEST });
        localStorage.removeItem(AUTH_TOKEN);
        setBag({});
    };

    useEffect(() => {
        const fetchData = async () => {
```

```
            dispatch({ type: SET_LOADING });

            let rawData;

            if (token) {
                try {
                    rawData = await axios({
                        method: 'GET',
                        url: PROFILE_ROUTE,
                        headers: {
                            Authorization: token,
                        },
                    });
                } catch (err) {
                    if (err.response.status === 401) {
                        console.log(err.message);
                    }
                }

                if (rawData.status === 200) {
                    const userData = rawData.data;

                    dispatch({ type: SET_LOGIN, payload: userData });
                }
                if (rawData.status === 401) {
                    localStorage.clear(AUTH_TOKEN);
                }
            } else {
                dispatch({ type: SET_GUEST });
            }
        };

        fetchData();
    }, [token]);

    appContext = {
        ...appContext,
        authData: {
            ...authState.authData,
            handleUserLogin,
            handleUserLogout,
        },
    };

    return (
        <>
            {!authState.isLoading && authState.authData && (
                <AppContext.Provider value={appContext}>
                    <Layout />
                </AppContext.Provider>
            )}
        </>
    );
}

ReactDOM.render(<App />, document.getElementById('root'));

serviceWorker.register();
```

Контейнери, що відповідають за відтворення компонентів:

Layout:

```
import React, { memo, useContext, useState } from 'react';
/* велика кількість імпортів компонентів, функцій, стилів, що не є
важливим для демонстрації принципу реалізації
*/

function Layout() {
    const classes = useStyles();
    const routeNameContext = useContext(RouteNameContext);
    const { bag, setBag } = useContext(AppContext);
    const [openBag, setOpenBag] = useState(false);
    const { isLoading, rawData } = useDataApi({
        url: `${CATEGORIES_ROUTE}`,
    });

    if (!isLoading && rawData) {
        const formated = rawData.reduce((prev, { id, name }) => {
            return {
                ...prev,
                [`${CATEGORIES_ROUTE}/${id}/products`]: name,
            };
        }, {});

        routeNameContext.addRoutes(formated);
    }

    const handleOpenBag = () => {
        setOpenBag(true);
    };

    const handleCloseBag = () => {
        setOpenBag(false);
    };

    const handleCancelItem = name => {
        setBag(prevBag => {
            const currentBag = { ...prevBag };

            delete currentBag[name];

            return currentBag;
        });
    };

    const setProductCount = name => ({ target }) => {
        const count = target.value > 1 ? target.value : 1;

        setBag(value => {
            const data = {
                ...value,
                [name]: {
                    ...value[name],
                    count,
                },
            };
            return data;
        });
    };

    return (
        <ThemeProvider theme={customizedTheme}>
            <BrowserRouter>
                {rawData && (
                    <Grid container className={classes.app}>
```

```
                                <Grid container direction="row">
                                    <Grid item className={classes.menu}>
                                        <Menu />
                                    </Grid>
                                    <Grid item className={classes.mainSection}>
                                        <Grid item>
                                            <SearchAppBar
                                                handleOpenBag={handleOpenBag}
                                            />
                                        </Grid>
                                        <Container
                                            maxWidth="lg"
                                            className={classes.ProductContainer}
                                        >
                                            <Grid item
classNameclasses.block}>
                                                <Switch>
                                                    <Route
                                                        exact

pathPRODUCTS_ROUTE}/:id`}
                                                    >
                                                        <ProductContainer
                                                            handleOpenBag={
                                                                handleOpenBag
                                                            }
                                                        />
                                                    </Route>
                                                    <Route
                                                        exact
                                                        path={

PRODUCTS_BY_CATEGORY_ROUTE
                                                        }
                                                    >
                                                        <ProductList />
                                                    </Route>
                                                    <Route exact
path={SIGNUP_ROUTE}>
                                                        <GoogleAuth />
                                                    </Route>
                                                    <Route exact
path={SIGNIN_ROUTE}>
                                                        <SignIn />
                                                    </Route>
                                                    <Route exact

path={PROFILE_ROUTE}>
                                                        <Profile />
                                                    </Route>
                                                    <Route exact
path={CHECKOUT_ROUTE}>
                                                        <Checkout />
                                                    </Route>
                                                </Switch>
                                            </Grid>
                                        </Container>
                                    </Grid>
                                </Grid>
                                <CssBaseline />
                            </Grid>
                        )}
                        <BagModal
                            bag={bag}
```

```
                    open={openBag}
                    onClose={handleCloseBag}
                    setProductCount={setProductCount}
                    handleCancelItem={handleCancelItem}
                />
            </BrowserRouter>
        </ThemeProvider>
    );
}

export default memo(Layout);
```

ВИСНОВОК

Було розглянуто підхід створення односторінкового веб-додатку для інтернет-торгівлі друкованими виданнями, методи його реалізації, проведено аналіз подібних ресурсів, побудовано інформаційну модель додатку, логічну схему реляційної бази даних, реалізовано програмно.

Технічний підхід односторінкового веб-додатку полягає у тому, що веб-сторінка динамічно завантажується. Тобто сторінка не перезавантажується повністю, а змінюються лише та інформація, яку запросив користувач. Це реалізується за допомогою того, що при переході на певну сторінку додатку завантажується JavaScript код, який створює на сторінці DOM, а всі дані, що пов’язані з певним параметром URL, завантажуються за допомогою AJAX – підходу для виконання асинхронного JavaScript. Це дозволяє уникати переривання досвіду користування між завантаженням контенту, що робить односторінковий веб-додаток більш швидким в порівнянні з веб-сайтом, що відправляє окремий HTML-документ у відповідь при переході по певному посиланню, тим самим роблячи веб-додаток подібним до настільного.

На стороні серверу використовується фреймворк, що реалізує шаблон проектування Модель-Вигляд-Контролер, який також реалізує шаблон Active Record, що значно спрощує спосіб маніпулювання даними у базі даних.

Отже, можна сказати, що створений інформаційний ресурс задовольняє всім визначеним вимогам.

СПИСОК ЛІТЕРАТУРИ

1. Володимир Дронов. Laravel. Быстрая разработка современных динамических Web-сайтов на PHP, MySQL, HTML и CSS. БХВ-Петербург, 2016 – 755 с.
2. От Web-сайтов к Web-приложениям: Часть 1. Web-сайт или Webприложение? [Електронний ресурс]. – Режим доступу: http://www.ibm.com/developerworks/ru/library/wa-websiteapp/
3. Початок роботи – React [Електронний ресурс]: React [Веб-сайт]. – Електронні дані. – Режим доступу: https://uk.reactjs.org/docs/getting-started.html
4. A JavaScript library for building user interfaces [Електронний ресурс]: React [Веб-сайт]. – Електронні дані. – Режим доступу: https://reactjs.org/
5. Daniel Bugl. Learn React Hooks: Build and refactor modern React.js applications using Hooks 1st Edition. Packt, 2019 – 426 с.
6. David Flanagan. JavaScript: The Definitive Guide, 7th Edition. NY: O'Reilly Media, 2020 – 462 с.
7. George Tillmann. Usage-Driven Database Design. Apress, 2017 – 398 с.
8. Installation – The PHP Framework for web artisans [Електронний ресурс]: Laravel [Веб-сайт]. – Електронні дані. – Режим доступу: https://laravel.com/docs/6.x/
9. Kelt Dockins. Design Patterns in PHP and Laravel. Apress, 2016. – 238 с.
10. Lockhart, Josh. Modern PHP: New Features and Good Practices. NY: O'Reilly Media, 2015. – 270 с.
11. Regina Obe, Leo Hsu. PostgreSQL: Up and Running, 3rd Edition. NY: O'Reilly Media, 2017 – 314 с.
12. Robin Nixon. Learning PHP, MySQL, JavaScript, CSS & HTML5, 3rd Edition. NY: O'Reilly Media, 2015 – 679 с.
13. Thakur, Dinesh. "What is a Database Server" interfaces [Електронний ресурс]: Ecomputernotes [Веб-сайт]. – Електронні дані. – Режим доступу:

https://ecomputernotes.com/fundamental/what-is-a-database/what-is-a-database-server#:~:text=A%20Database%20Server%20is%20a,them%20back%20over%20the%20network.

14. The Single Page Interface Manifesto [Електронний ресурс]: Source Forge [Веб-сайт]. – Електронні дані. – Режим доступу: http://itsnat.sourceforge.net/php/spim/spi_manifesto_en.php
15. What Is Load Balancing? How Load Balancers Work [Електронний ресурс]: NGINX [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.nginx.com/resources/glossary/load-balancing