МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ

Кафедра комп’ютерних наук та кібербезпеки

СЮХ ОЛЕКСАНДР МИКОЛАЙОВИЧ

ПРОЄКТУВАННЯ ТА РОЗРОБКА КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ ОНЛАЙН-ТОРГІВЛІ З ПІДХОДОМ ОДНОСТОРІНКОВОЇ АРХІТЕКТУРИ

Спеціальність: 122 Комп’ютерні науки Освітньо-професійна програма: Комп’ютерні науки та інформаційні технології Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:
Чернящук Наталія Леонідівна,
професор кафедри комп’ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол №_____
засідання кафедри комп’ютерних наук
та кібербезпеки
від _____ 2025 р.
Завідувач кафедри
(_____) Гришанович Т. О.

ЛУЦЬК 2025

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних умовах цифровізації торгівлі особливого значення набуває створення ефективних вебсистем для здійснення онлайн-продажів. Одним із перспективних рішень є застосування односторінкової архітектури (SPA), що забезпечує високу швидкодію, інтерактивність та зручність користування. Зростання обсягів електронної комерції та зміщення споживчої поведінки в онлайн-середовище зумовлюють потребу у створенні зручних і швидкодійних вебзастосунків. Традиційні багатосторінкові сайти поступово передають місце односторінковим застосункам, які забезпечують кращий користувацький досвід завдяки динамічному оновленню вмісту без повного перезавантаження сторінки. Такий підхід дозволяє значно зменшити затримки при взаємодії з інтерфейсом, оптимізувати навантаження на сервер і підвищити продуктивність вебсистеми. У цьому контексті актуальним є проєктування і впровадження сучасних архітектурних рішень для клієнт-серверних систем, що відповідають високим вимогам до масштабованості, безпеки та зручності використання в сфері онлайн-торгівлі.

Метою кваліфікаційної роботи є проєктування та розробка клієнт-серверної вебсистеми онлайн-торгівлі з односторінковою архітектурою, що забезпечує інтерактивний інтерфейс для користувача, ефективну обробку даних на стороні сервера та підтримку типових функцій електронної комерції.

Для досягнення поставленої мети в межах дослідження передбачено реалізацію **таких завдань:**

- проаналізувати особливості клієнт-серверних систем у сфері електронної комерції;
- визначити вимоги до функціональності та архітектури вебзастосунку;
- спроєктувати структуру клієнт-серверної системи та бази даних відповідно до предметної області;
- реалізувати серверну частину з підтримкою REST API для обробки запитів;

- впровадити механізми аутентифікації та авторизації користувачів;
- розробити клієнтську частину з використанням односторінкової архітектури;
- забезпечити інтеграцію між клієнтською та серверною частинами;
- здійснити тестування системи на предмет коректності роботи, продуктивності та масштабованості.

Об’єктом дослідження є клієнт-серверні інформаційні системи електронної комерції, зокрема їх архітектура, користувацький інтерфейс та функціональні компоненти, що забезпечують обробку запитів, керування даними та взаємодію з користувачем у вебсередовищі.

Предметом дослідження є процес проєктування, розробки та функціонування клієнт-серверної вебсистеми з односторінковою архітектурою, зокрема: застосування принципів компонентно-орієнтованої розробки інтерфейсу, організація серверної логіки для роботи з базою даних, а також дослідження способів підвищення продуктивності та масштабованості програмного продукту.

Публікації

У ході виконання кваліфікаційної роботи підготовлено та опубліковано тези доповіді в межах міжнародної науково-практичної конференції “Проблеми комп’ютерних наук, програмного моделювання та безпеки цифрових систем”:

1. Сюх О. М. Методи оптимізації продуктивності односторінкових застосунків на прикладі фреймворку React. Проблеми комп’ютерних наук, програмного моделювання та безпеки цифрових систем.: матеріали II міжнар. науково-практ. конф. Луцьк, 9–11 червн. 2025 р. Луцьк, 2025. С. 57—58. URL : https://apcssm.vnu.edu.ua/index.php/conf/article/view/198

РОЗДІЛ 1.
АНАЛІЗ ТА ОСОБЛИВОСТІ ОДНОСТОРІНКОВИХ СИСТЕМ ОНЛАЙН-ПРОДАЖУ

1.1. Розвиток односторінкової архітектури вебдодатків

У сучасній веброзробці значну популярність здобули односторінкові вебдодатки (Single Page Applications, SPA), які забезпечують зручний, швидкий і динамічний користувацький досвід. Розвиток SPA-архітектури відбувався як відповідь на обмеження традиційних багатосторінкових підходів (Multi Page Applications, MPA), а також під впливом технологічного прогресу у сфері фронтенд-розробки. Цей тип архітектури трансформував підхід до створення вебінтерфейсів, поєднавши у собі ефективність, інтерактивність та масштабованість [3].

Перші вебсайти будувалися за принципом класичної багатосторінкової архітектури, де кожен перехід між сторінками вимагав повного перезавантаження вмісту. Такий підхід був достатнім у період, коли обсяг інформації був невеликим, а інтернет-з’єднання — повільним. Однак із розвитком вебтехнологій та підвищенням очікувань користувачів щодо швидкодії, інтуїтивності та зручності взаємодії виникла потреба в нових рішеннях.

Поява AJAX — технології, яка дозволяє вебдодаткам обмінюватися даними з сервером у фоновому режимі, на початку 2000-х років стала поворотним моментом. Вона дозволила оновлювати частини сторінки без повного перезавантаження, що значно покращувало користувацький досвід. Саме на основі цієї ідеї почали формуватися перші концепції односторінкових вебдодатків, де взаємодія з сервером відбувається асинхронно, а навігація здійснюється в межах однієї сторінки. Крім того, поширення мобільних пристроїв та поява потужних фреймворків сприяли переходу до більш гнучкої та інтерактивної SPA-архітектури.

Початковий етап характеризувався використанням бібліотек у поєднанні з технологією AJAX. Завдяки цьому стало можливим оновлювати частини вебсторінки без повного її перезавантаження, що значно покращило користувацький досвід.

Наступним кроком стало впровадження MVC-фреймворків, які вперше надали структуру для побудови складніших клієнтських додатків. Ці інструменти дозволили краще організовувати код, поділяючи його на логічно ізольовані частини: моделі, представлення та контролери, що зробило розробку більш системною та масштабованою.

Далі почав домінувати компонентний підхід, який ґрунтується на розбитті інтерфейсу на незалежні багаторазові елементи — компоненти. У цьому напрямі вирізняються такі рішення, як React, Vue та оновлений Angular — фреймворки та бібліотеки для побудови інтерфейсів користувача, що спеціалізуються на створенні динамічних односторінкових застосунків. Компонентна архітектура дала змогу ефективніше управляти станом додатка, повторно використовувати частини інтерфейсу та створювати складні SPA із зручною структурою [5].

Одним із найбільш впливових рішень став React — бібліотека з відкритим кодом, із запровадженою концепцією віртуального *DOM*. Цей механізм дозволяє ефективно оновлювати лише ті частини інтерфейсу, які змінилися, що значно підвищує продуктивність застосунків. Крім того, React ввів сучасну модель управління станом через хуки — функції, які дозволяють підключати логіку до функціональних компонентів, що спростило структуру додатків і зробило її більш гнучкою [1].

Angular, на відміну від React, є повноцінним фреймворком, який надає розробникам усе необхідне для створення масштабних застосунків. Він підтримує впровадження залежностей, вбудовану роутизацію, реактивні форми, механізми валідації, а також засоби тестування та розгортання. Завдяки цьому Angular застосовується переважно в корпоративних рішеннях, де важлива чітка структура і сувора типізація.

Vue поєднує в собі ідеї React та Angular, але при цьому вирізняється простотою і гнучкістю. Завдяки низькому порогу входу, Vue є зручним для швидкого старту невеликих проєктів, але водночас надає інструменти для побудови повноцінних SPA, якщо того вимагає проєкт.

Розвиток SPA-архітектури також був би неможливим без появи спеціалізованих інструментів збірки та трансформації коду, таких як Webpack, Babel та Vite. Вони забезпечують обробку сучасного JavaScript (ES6+) - ключової мови програмування у розробці клієнтської частини вебзастосунків. А також: оптимізацію продуктивності, динамічне імпортування модулів і гаряче оновлення модулів у процесі розробки. Зокрема, Vite, завдяки використанню нативних можливостей браузера значно скорочує час запуску проєкту та пришвидшує розробку, що особливо важливо для великих SPA.

Таким чином, сучасні JavaScript-фреймворки та супутні інструменти створили умови для гнучкого, масштабованого та ефективного проєктування односторінкових вебдодатків, адаптованих до вимог сучасної веброзробки.

1.2.Особливості односторінкових систем онлайн-торгівлі

1.2.1. Принципи роботи SPA

Односторінковий застосунок (SPA, Single Page Application) є сучасним підходом до побудови вебархітектури, що передбачає завантаження лише одного HTML-документа, в межах якого динамічно оновлюється вміст за потребою користувача. У SPA взаємодія з користувачем відбувається без повного перезавантаження сторінки, що забезпечує плавніший та швидший досвід використання.

На відміну від традиційних багатосторінкових застосунків, де кожен перехід між сторінками супроводжується запитом до сервера та завантаженням нової HTML-сторінки, у SPA зміна вмісту реалізується засобами JavaScript без оновлення всієї сторінки. Це дозволяє зменшити кількість мережевих запитів і підвищити швидкодію, особливо при повторному відвідуванні сторінок або

здійсненні частих дій користувача.

JavaScript відіграє ключову роль у побудові односторінкових вебзастосунків, забезпечуючи інтерактивність, динамічне оновлення інтерфейсу та взаємодію з сервером. На відміну від традиційних сайтів, де структура й вміст сторінки формуються на сервері, у SPA основна відповідальність за рендеринг і логіку переходить на клієнтську сторону, де JavaScript виконує центральну функцію [4].

Рендеринг інтерфейсу в SPA реалізується за допомогою JavaScript-фреймворків та бібліотек, які дозволяють створювати компоненти — незалежні частини інтерфейсу, що можуть оновлюватися окремо без перезавантаження всієї сторінки. Це дає змогу динамічно змінювати вміст DOM (Document Object Model) у відповідь на дії користувача, забезпечуючи високий рівень інтерактивності.

Окрім побудови інтерфейсу, JavaScript відповідає за реалізацію бізнес-логіки застосунку. Він обробляє події, керує станом інтерфейсу, здійснює валідацію даних та координує взаємодію між різними частинами програми, а також комунікує із сервером. Завдяки цьому користувачі можуть взаємодіяти з додатком майже так само, як із нативними програмами, що працюють локально.

Однією з ключових особливостей односторінкових вебзастосунків є реалізація маршрутизації на клієнтському боці. Це означає, що навігація між візуальними станами додатка відбувається миттєво, що значно покращує взаємодію користувача з інтерфейсом.

Завдяки сучасним можливостям браузерів, таким як історія запитів, односторінкові застосунки мають змогу змінювати адресу, не оновлюючи сторінку повністю. Це забезпечує більш плавний досвід навігації, збереження історії перегляду, а також полегшує реалізацію логіки переходів у межах складного інтерфейсу.

В архітектурі односторінкових застосунків (SPA) важливу роль відіграє динамічне завантаження даних у процесі взаємодії з інтерфейсом. На відміну від традиційних багатосторінкових рішень, де зміна інформації передбачає

повне оновлення сторінки, SPA працюють із даними безперервно, підвантажуючи їх у відповідь на конкретні дії користувача.

Для цього застосунок звертається до зовнішнього або внутрішнього API (Application Programming Interface), надсилаючи асинхронні запити. У відповідь сервер надає лише необхідні фрагменти інформації, які інтегруються в інтерфейс без повного перезавантаження сторінки.

Управління станом є однією з центральних задач у розробці односторінкових застосунків (SPA), оскільки такі додатки активно взаємодіють з користувачем та динамічно змінюють інтерфейс без повного оновлення сторінки. Стан зберігає сукупність даних, що описують поточне положення застосунку.

У невеликих застосунках стан можна зберігати локально в окремих компонентах, однак із зростанням складності виникає потреба в централізованому підході до управління. Саме для цього використовуються інструменти на кшталт React Context, Redux - бібліотеки управління станом. Вони дозволяють зручно передавати дані між різними частинами інтерфейсу, підтримувати їхню узгодженість та реагувати на зміни в реальному часі.

Окрім зберігання, важливим аспектом є синхронізація стану із зовнішніми даними — API запитами, на основі відповіді яких оновлюється інтерфейс відповідно до нових значень.

У SPA-застосунках важливу роль відіграє можливість зберігати дані безпосередньо на клієнтському боці. Це дозволяє зменшити кількість запитів до сервера, пришвидшити завантаження інтерфейсу та забезпечити збереження деяких даних між сеансами користувача. У цьому контексті найбільш поширеними технологіями є localStorage та sessionStorage.

За допомогою localStorage можна забезпечувати довготривале збереження пар даних у локальній пам’яті браузера. Його часто використовують для зберігання не чутливої інформації: мовних налаштувань, теми інтерфейсу або ідентифікаторів сесій. Подібно працює sessionStorage, але дані обмежуються поточним сеансом, після чого видаляються з пам’яті браузера.

Односторінкові вебзастосунки (SPA) являють собою сучасний підхід до розробки інтерфейсів, орієнтований на високу швидкодію, інтерактивність і зручність користування. Завдяки активному використанню JavaScript, клієнтської маршрутизації, динамічного завантаження даних, централізованого управління станом і можливостей кешування, SPA забезпечують наближений до нативного досвід взаємодії з вебінтерфейсом, що робить їх одними з найпоширеніших рішень у сфері онлайн-торгівлі та веброзробки загалом [20].

1.2.2. Переваги та недоліки SPA у сфері онлайн-торгівлі

У сучасній онлайн-торгівлі швидкість і зручність взаємодії користувача із вебзастосунком мають вирішальне значення. Саме тому архітектура SPA набуває все більшого поширення серед інтернет-магазинів. Вона дозволяє створювати гнучкі, динамічні та зручні у використанні рішення, проте має також і потенційні недоліки, які потрібно додатково вирішувати.

Переваги. Одна з ключових переваг SPA (Single Page Application) у сфері онлайн-торгівлі — це суттєве покращення користувацького досвіду (UX). Завдяки тому, що SPA завантажує єдину HTML-сторінку і динамічно оновлює її вміст без повного перезавантаження, взаємодія з сайтом стає швидкою та плавною. Користувач може миттєво переходити між категоріями, товарами чи сторінками кошика без затримок, викликаних повторним завантаженням сторінок із сервера. Це скорочує час очікування та підвищує рівень задоволення від користування ресурсом, що, у свою чергу, позитивно впливає на поведінкові фактори, тривалість перебування на сайті та ймовірність здійснення покупки.

Ще однією важливою перевагою SPA у сфері онлайн-торгівлі є підвищена швидкість роботи та загальна продуктивність застосунку. Після початкового завантаження всі основні ресурси вже перебувають у браузері, і подальші дії користувача, як-от перегляд товару чи фільтрація за категоріями, відбуваються без додаткових повних запитів до сервера. Це не лише прискорює відображення нового вмісту, а й зменшує навантаження на серверну

інфраструктуру. Завдяки кешуванню та повторному використанню інтерфейсних компонентів SPA дозволяє ефективніше управляти ресурсами, що є особливо цінним для великих інтернет-магазинів із високим трафіком.

Висока сумісність із мобільними пристроями та можливість інтеграції в прогресивні вебдодатки є ще однією перевагою односторінковості. Такий підхід дозволяє створювати застосунки, що адаптивно працюють на різних екранах і забезпечують користувачам зручний інтерфейс, подібний до нативних мобільних програм. Крім того, може працювати навіть без постійного підключення до Інтернету — сторінки й дані кешуються у браузері, що забезпечує базову функціональність у режимі офлайн. Це особливо актуально для користувачів, які переглядають товари в умовах нестабільного з'єднання.

Недоліки. Одним із ключових недоліків використання SPA в онлайн-торгівлі є складнощі з пошуковою оптимізацією (SEO). Оскільки вміст односторінкового застосунку генерується динамічно за допомогою JavaScript, пошуковим роботам може бути складно коректно індексувати окремі сторінки, зокрема сторінки товарів чи категорій. Це може негативно вплинути на видимість інтернет-магазину в результатах пошуку. Для вирішення цієї проблеми часто необхідно впроваджувати серверний рендеринг (SSR) або пререндеринг, що потребує додаткових ресурсів і ускладнює архітектуру застосунку.

Ще одним недоліком SPA у сфері онлайн-торгівлі є тривалий початковий час завантаження. Через необхідність завантаження великого обсягу збірки коду, перший візит користувача на сайт може супроводжуватися помітною затримкою — особливо за умов повільного або нестабільного інтернет-з'єднання. Це може призвести до втрати частини потенційних клієнтів ще до того, як вони побачать головну сторінку магазину, що критично для комерційних ресурсів, де швидкість доступу напряму впливає на конверсію. Частково цю проблему можна вирішити за допомогою умовного рендерингу або розділення коду, що дозволяє завантажувати лише ті частини застосунку, які необхідні на поточному етапі.

Ще одним важливим аспектом, який потребує уваги при використанні SPA у сфері онлайн-торгівлі, є безпека. Оскільки основна логіка SPA виконується на клієнтській стороні, застосунок стає більш вразливим до атак, при яких зловмисники можуть впровадити шкідливий код у вебсторінку. Крім того, всі API-запити, через які відбувається обмін даними з сервером, повинні бути належним чином захищені — із використанням авторизації, аутентифікації та перевірки прав доступу. У контексті онлайн-торгівлі це особливо критично, адже йдеться про обробку конфіденційної інформації користувачів, зокрема особистих даних та платіжних реквізитів.

Загалом, застосування архітектури SPA у сфері онлайн-торгівлі має як суттєві переваги, так і виклики, що потребують уважного опрацювання. З одного боку, SPA забезпечує швидку, зручну та інтерактивну взаємодію з користувачем, покращує продуктивність застосунку, знижує навантаження на сервер та сприяє кращій мобільній адаптації. З іншого боку, існують обмеження, пов’язані з SEO, безпекою та тривалим початковим завантаженням, які можуть вплинути на ефективність бізнесу. Тому успішне впровадження SPA в онлайн-торгівлі передбачає збалансований підхід, який враховує як технічні переваги, так і потенційні ризики, із впровадженням відповідних рішень для їх мінімізації.

1.3. Архітектурні підходи до створення односторінкових вебдодатків

1.3.1. Архітектура клієнтської частини системи

Клієнтська частина односторінкового вебзастосунку є критично важливою складовою загальної архітектури, оскільки саме вона відповідає за рендеринг інтерфейсу користувача, обробку взаємодії та динамічну зміну вмісту без перезавантаження сторінки. Одним із ключових підходів до побудови SPA є компонентно-орієнтована архітектура, яка передбачає поділ інтерфейсу на незалежні, багаторазово використовувані частини — компоненти. Кожен компонент відповідає за окрему логіку чи елемент

інтерфейсу, наприклад, картку товару, форму пошуку або навігаційне меню. Крім компонентів, застосунок структурується також на сервіси, утиліти, сторінки, що дозволяє розділити відповідальність і спростити супровід та масштабування коду. Такий модульний підхід забезпечує кращу підтримуваність, повторне використання та тестованість клієнтського коду. На рисунку 1.1 представлено узагальнену структуру клієнтської частини SPA-застосунку, що ілюструє взаємозв'язки між основними складовими.

Рисунок 1.1 – Структура клієнтської частини spa-застосунку

Важливим елементом клієнтської архітектури є маршрутизація, яка дозволяє реалізувати навігацію між різними віртуальними сторінками без повного перезавантаження браузера. У SPA це зазвичай реалізується за допомогою бібліотек. Вони забезпечують внутрішню маршрутизацію, обробку параметрів у адресі, динамічне підвантаження вмісту на основі маршруту, а також обробку помилок. Завдяки такому підходу користувач отримує досвід,

подібний до традиційних десктопних застосунків.

Управління станом застосунку також є невід'ємною частиною архітектури SPA. У невеликих застосунках достатньо використання локального стану, однак у масштабніших проектах виникає потреба у глобальному сховищі, де зручно централізовано зберігати дані, доступні багатьом компонентам одночасно. Для цього застосовуються спеціалізовані бібліотеки, зокрема Redux. Крім того, для збереження інформації між сесіями можуть використовуватись механізми локального сховища, такі як localeStorage та sessionStorage.

Ще одним важливим аспектом клієнтської архітектури є взаємодія з сервером, яка реалізується через REST API — інтерфейс прикладного програмування, що дозволяє обмінюватися даними між клієнтом і сервером у структурованому форматі. Застосунок надсилає запити до серверної частини з метою отримання чи оновлення даних. Для виконання таких запитів застосовуються бібліотеки Axios, Fetch API або аналоги. У SPA важливо реалізувати обробку помилок, повторні спроби запитів у разі збоїв, а також стратегії кешування, що дозволяють зменшити кількість звернень до сервера й підвищити швидкість роботи. У складніших сценаріях використовуються механізми автоматичного оновлення стану після зміни на сервері.

Усі зазначені аспекти клієнтської архітектури SPA тісно взаємопов'язані та впливають на зручність, масштабованість і стабільність роботи застосунку. Вибір конкретних рішень у кожному з цих напрямів має базуватись на особливостях проєкту, цільовій аудиторії та вимогах до продуктивності й безпеки. Комплексне врахування компонентної структури, ефективної маршрутизації, керування станом і взаємодії з сервером дозволяє побудувати сучасний, надійний та продуктивний SPA-застосунок.

1.3.2. Архітектура серверної частини системи

Серверна частина вебдодатку відіграє ключову роль у забезпеченні функціональності, безпеки та масштабованості системи, особливо у сфері

онлайн-торгівлі. Вона відповідає за обробку запитів від клієнтської частини (SPA), збереження та обробку даних, взаємодію з базою даних, реалізацію бізнес-логіки, а також за безпечну комунікацію між усіма компонентами. Оптимальна архітектура серверної частини (див. рис. 1.2) серверної частини дозволяє досягнути високої продуктивності системи, забезпечити підтримку великої кількості користувачів та гарантувати стабільну роботу вебдодатку.

Рисунок 1.2 – Структура серверної частини

Однією з основ серверної частини є створення REST API — інтерфейсу прикладного програмування. Такий API надає уніфікований спосіб взаємодії між клієнтом та сервером, де кожен ресурс представлений у вигляді URL-адреси, а основні операції виконуються через HTTP-методи. Для коректної взаємодії з клієнтською частиною важливо реалізувати обробку помилок із поверненням відповідних кодів статусів, що дозволяє SPA застосунку реагувати на події на стороні сервера передбачуваним чином [18].

Наступним критично важливим аспектом є серверна обробка даних, яка включає реалізацію бізнес-логіки, валідацію вхідної інформації та ефективну обробку великих обсягів даних. Перед обробкою запиту необхідно перевірити дані, що надходять від клієнта, наприклад, за допомогою бібліотек Joi або Yup, що знижує ризик помилок та зловживань. Також поширеною практикою є реалізація фільтрації, сортування та пагінації для списків продуктів, що дозволяє користувачеві зручно працювати з великими каталогами товарів. На рівні бізнес-логіки здійснюються обчислення, наприклад, визначення підсумкової вартості замовлення з урахуванням знижок, доставки або податків.

Для побудови гнучкої та підтримуваної серверної частини доцільно застосовувати перевірені архітектурні шаблони, зокрема багаторівневу архітектуру. У ній сервер поділений на окремі шари: контролери, які обробляють HTTP-запити; сервіси, що містять бізнес-логіку; репозиторії, які взаємодіють з базою даних. Такий підхід сприяє модульності та легшій підтримці коду. Окрім того, широко використовуються middleware-компоненти — програмні проміжні шари, що обробляють запити ще до їх потрапляння у бізнес-логіку.

Окрему увагу необхідно приділити безпеці серверної частини. Для захисту API від несанкціонованого доступу застосовуються механізми аутентифікації та авторизації — зокрема, JSON Web Tokens (JWT) або OAuth2. Така реалізація дозволяє перевіряти права доступу до певних ресурсів відповідно до ролі користувача. Також важливо коректно налаштувати політику CORS (Cross-Origin Resource Sharing), щоб обмежити небажані джерела, з яких можуть надходити запити до серверного API.

Отже, архітектура серверної частини вебдодатку повинна враховувати як функціональні вимоги системи, так і аспекти безпеки, масштабованості та підтримуваності. Грамотне розділення обов’язків між компонентами, дотримання стандартів REST, реалізація бізнес-логіки та захист API створюють надійний фундамент для ефективної взаємодії з клієнтською SPA-частиною в системах онлайн-торгівлі.

1.3.3. Забезпечення безпеки та аутентифікація у SPA додатках

Забезпечення безпеки та реалізація аутентифікації у односторінкових додатках (SPA) є критично важливими аспектами архітектури, що безпосередньо впливають на надійність, стійкість до атак і загальну захищеність вебсистеми. Оскільки SPA функціонує як повноцінний клієнт, який активно взаємодіє з сервером через API, особливу увагу слід приділяти виявленню вразливостей, захисту даних і правильному впровадженню механізмів автентифікації та авторизації.

У сучасних SPA-додатках застосовують два основних підходи до аутентифікації: сесійну (через cookie) та на основі токенів. Сесійна аутентифікація передбачає збереження ідентифікатора сесії на сервері та передачу його у cookie з кожним запитом. Безсесійна, або токен-орієнтована аутентифікація, використовує токени, які генеруються після успішної ідентифікації користувача. Порівняння обох методів показує, що токен-орієнтований підхід краще підходить для SPA, оскільки не потребує серверного збереження сесій і краще масштабується [10].

JSON Web Token є найпоширенішим форматом у безсесійній аутентифікації. Він складається з трьох частин: header, payload та signature. Після генерації токена на сервері, він передається клієнту та надалі використовується для автентифікації через заголовок авторизації. Важливо реалізувати оновлення access-токенів через refresh-токени, які дозволяють продовжити сесію без повторного входу. Такий підхід підвищує зручність і безпеку користувача.

Авторизація полягає у перевірці прав доступу користувача до певних ресурсів. Для цього часто впроваджується модель ролей: адміністратор, звичайний користувач, гість тощо. На сервері перевіряються права доступу до API залежно від ролі, тоді як у клієнтській частині реалізуються механізми обмеження доступу до маршрутів. Це дозволяє приховувати інтерфейсні елементи або сторінки від неавторизованих користувачів. На рисунку 1.3 представлено узагальнену схему архітектури безпеки та автентифікації у SPA-

додатках, яка ілюструє основні компоненти взаємодії між клієнтом і сервером, використання JWT, механізми авторизації та захисту API.

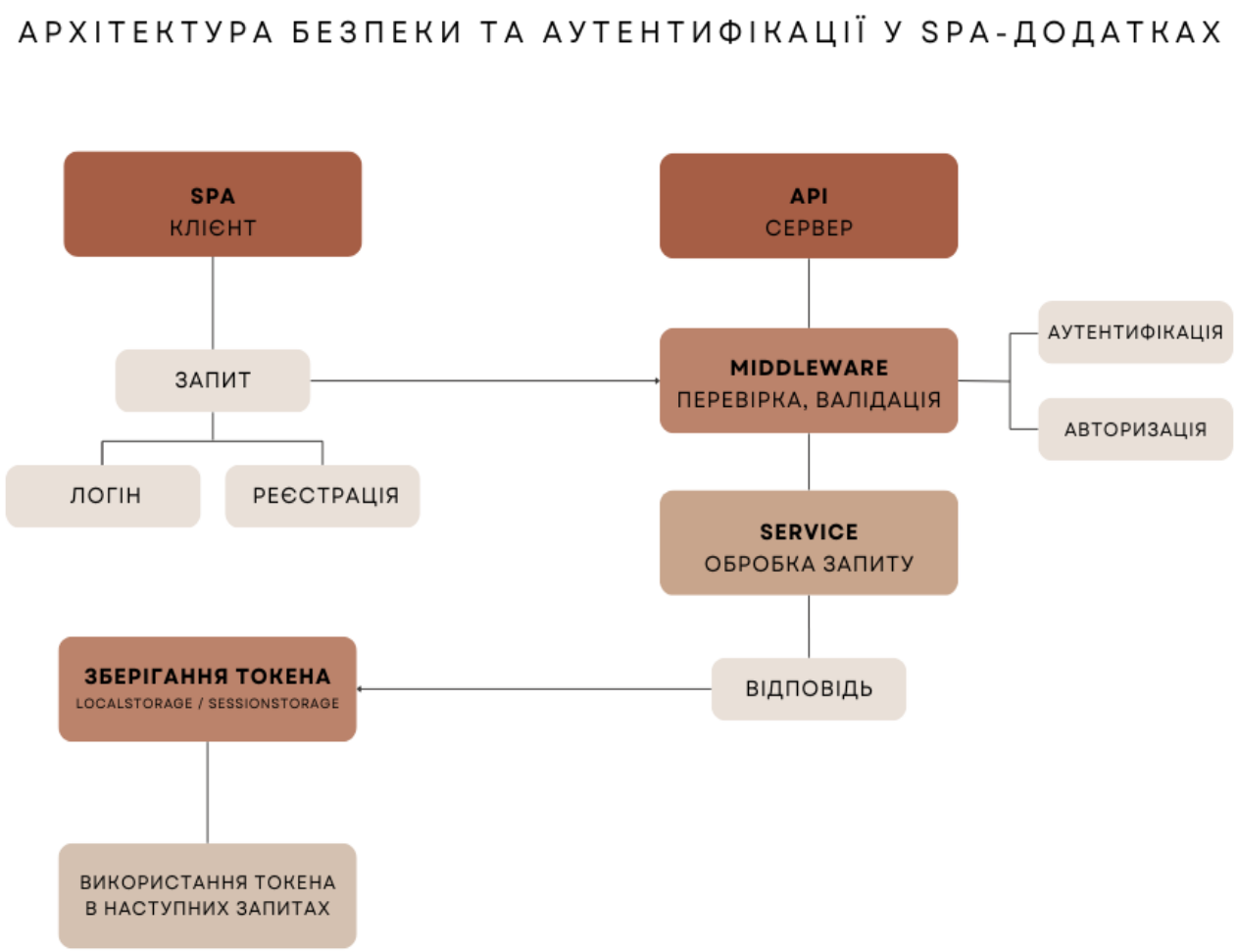

Рисунок 1.3 - Архітектура безпеки та аутентифікації і SPA-додатках

Захист API — це критично важлива частина архітектури SPA. Сервер повинен перевіряти валідність токена в middleware, обробляти неавторизовані запити відповідними статусами. У SPA особливо актуальною є CORS — політика, що забороняє робити запити із недозволеного домену, оскільки фронтенд та бекенд часто працюють на різних доменах або портах. Сервер має чітко вказувати дозволені джерела, методи та заголовки у відповіді.

У підсумку, безпека та аутентифікація є невід’ємною частиною архітектури SPA-додатків. Вибір відповідних технологій, дотримання принципів розмежування доступу та впровадження захисту як на клієнтському,

так і на серверному рівні, дозволяють створити захищене середовище для користувачів та забезпечити стабільну роботу додатку у виробничих умовах.

1.4. Огляд існуючих систем онлайн-продажу та їхніх функціональних можливостей

На даний момент існує безліч онлайн-магазинів, оскільки продаж товарів через інтернет стає все більш популярним, для аналізу були відібрані вебсайти, що спеціалізуються на продажі одного типу товару:

- Artspace SOL`;
- Книгарня Є;
- GymBeam.

Artspace SOL (рис. 1.4) – це вебдодаток для купівлі листівок та посуду з авторськими ілюстраціями розроблений компанією “Artspacesol”.

Рисунок 1.4 – Artspace SOL`

Онлайн-магазин був розроблений за допомогою фреймворку React на базі JavaScript та обов’язкових HTML5 та CSS3, даний програмний продукт виконує наступні функції:

- показ каталогу товарів за категоріями та тематиками (рис. 1.5);
- пошук товарів за ключовими словами;

- інтерактивний кошик з можливістю редагувати властивості товарів;
- опція створення персонального дизайну ілюстрації для клієнтів;
- інтеграція з платіжними системами для онлайн-оплати.

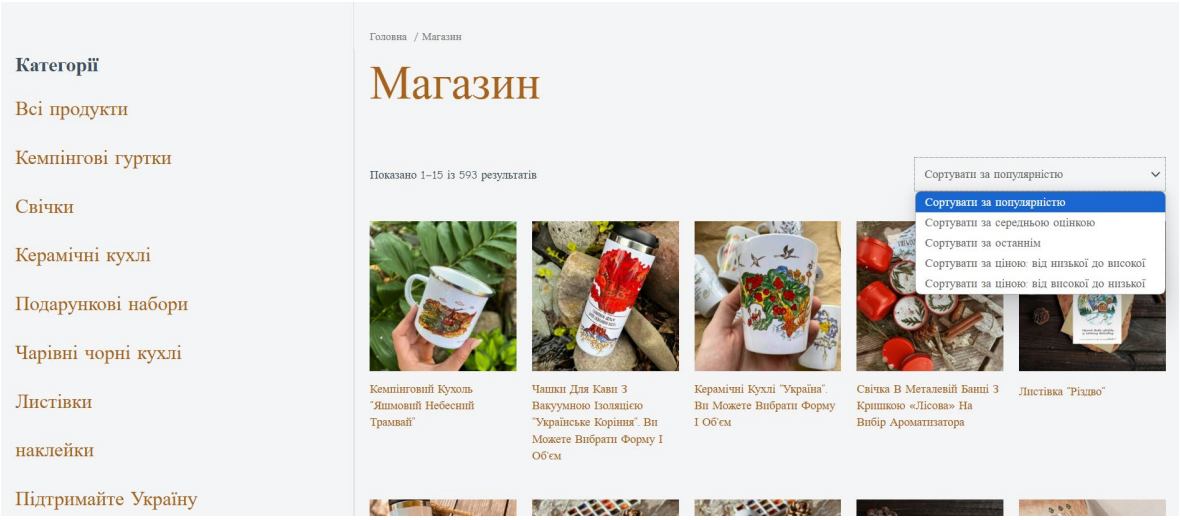

Рисунок 1.5 – Каталог вебдодатку Artspace SOL`

Клієнтоорієнтований інтерфейс сприяє зацікавленості користувача в продукті, а завдяки оптимізованому коду користувач не має проблем із затримками в роботі сайту. Відсутність багатомовності вебдодатку дещо ускладнює користування пересічного користувача, проте із використанням сучасних веббраузерів цей недолік не є критичним.

Книгарня Є (рис. 1.6) – це вебдодаток розроблений австрійською компанією “ECEM Media GmbH” та побудований за принципом клієнт-сервер. Основними програмними інструментами про розробці є: PHP (гіпертекстовий препроцесор для створення HTML-сторінок), mySQL (мова програмування, що дозволяє маніпулювати базами даних) та HTML/CSS.

Програмна версія книгарні виконує наступний функціонал:

- з першої сторінки закликає користувача до купівлі найпопулярніших продуктів за допомогою різних інтерактивних компонентів (реклами, знижки тощо) (рис. 1.7);
- модуль відгуків для кожного продукту та їх статистика;
- особистий кабінет для клієнтів з персонально підібраними жанрами та

знижками на товари;

- функція збирання балів, що нараховуються за купівлі продукту компанії;
- інтеграція з кур'єрськими службами та платіжними системами для онлайн-оплати.

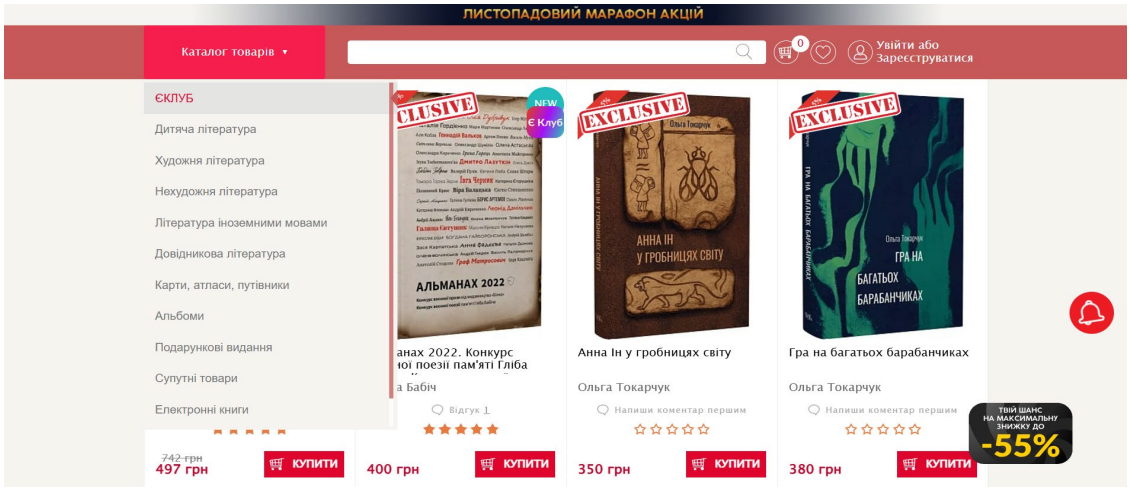

Рисунок 1.6 – Система сайту Книгарня Є

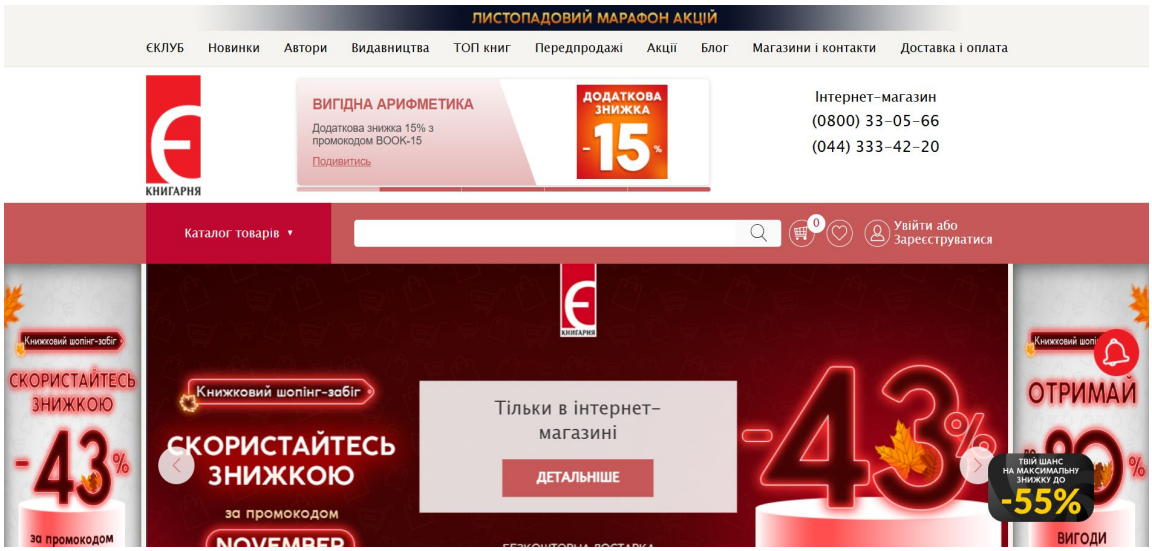

Рисунок 1.7 – Клієнтоорієнтована частина вебдодатку

Сайтом може користуватись пересічний користувач через його багатомовність та відносно легку навігацію. Натомість через великий обсяг даних, зокрема фото, сайт повільно завантажується без попереднього кешування браузером, що свідчить про недостатню оптимізацію програмної

розробки.

GymBeam (рис. 1.8) – це вебдодаток розроблений однойменною компанією для онлайн-купівлі спортивних товарів. За допомогою локальної пам’яті веббраузеру реалізовано показ рекомендацій на основі попередніх покупок. Виконано опцію підписки на регулярну доставку товарів, а також інтеграцію з популярними платіжними системами та кур’єрськими компаніями.

Програмний продукт має повністю адаптивний інтерфейс для мобільних та планшетних пристроїв, розширені можливості аналітики для клієнтів та високу швидкість завантаження сторінок, що свідчить про оптимізацію кожної частини вебдодатку. Недоліком є дещо складний інтерфейс для пересічного користувача.

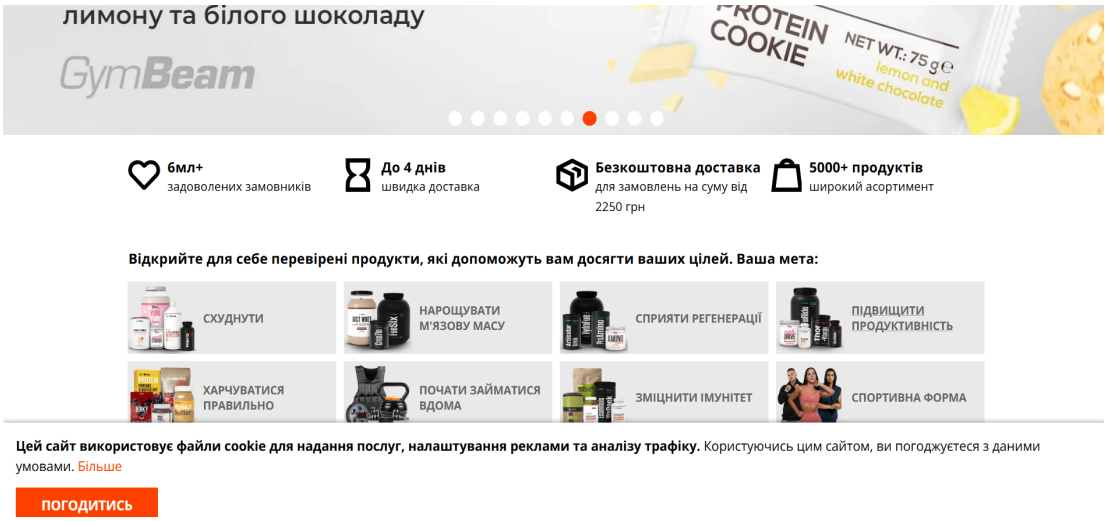

Рисунок 1.8 – Вебдодаток GymBeam

Аналіз існуючих рішень демонструє, що програмні розробки мають як переваги, так і обмеження. Artspace SOL є прикладом сучасного підходу з використанням передових вебтехнологій. Книгарня Є орієнтується на локального користувача, проте потребує значної оптимізації для швидшої роботи. GymBeam показує приклад інноваційних функцій, проте його інтерфейс може відлякати нових клієнтів. Усі ці особливості враховані при проектуванні онлайн-магазину, щоб уникнути недоліків і підкреслити найкращі риси.

РОЗДІЛ 2.
ПРОЄКТУВАННЯ ТА РОЗРОБКА КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ ОНЛАЙН-ТОРГІВЛІ

2.1. Постановка задачі, призначення та вимоги до програмного засобу

Постановка задачі

У рамках даного проєкту розробляється клієнт-серверна система онлайн-торгівлі, основною метою якої є забезпечення користувачів зручним та інтуїтивно зрозумілим інтерфейсом для перегляду, пошуку, замовлення та придбання наклейок різного призначення. Система має бути доступною через сучасні веббраузери та підтримувати адаптивну верстку для коректного відображення на мобільних пристроях. Крім цього, програмний засіб передбачає реалізацію функціональних можливостей для адміністрування товарного каталогу, включаючи додавання, редагування та видалення позицій, а також обробку замовлень і формування базової аналітики щодо динаміки продажів. Проєктування такої системи передбачає побудову як клієнтської, так і серверної частин з чітко визначеними зонами відповідальності, а саме: клієнтська частина відповідає за взаємодію з користувачем і візуальне відображення даних, тоді як серверна частина виконує функції обробки запитів, управління даними та забезпечення бізнес-логіки.

Призначення системи. Програмний засіб орієнтований на задоволення потреб як кінцевих користувачів, так і адміністратора онлайн-магазину. Для користувачів реалізується можливість створення облікового запису шляхом реєстрації, подальшого входу до персонального кабінету, а також здійснення основних дій, пов’язаних з онлайн-покупками: перегляд карток товарів з детальною інформацією, пошук наклейок за категоріями або ключовими словами, додавання товарів до кошика, редагування вмісту кошика та формування і підтвердження замовлення. Серверна частина обробляє всі запити, що надходять від клієнта, забезпечує зв’язок з базою даних, у якій

зберігаються відомості про користувачів, товари та замовлення, а також реалізує критично важливі механізми безпеки — автентифікацію та авторизацію. Окрім того, система надає адміністративний інтерфейс, що дозволяє керувати асортиментом товарів, контролювати замовлення та оновлювати інформацію про наявність і ціни.

Вимоги до системи. До клієнт-серверної системи онлайн-торгівлі висувається низка функціональних і нефункціональних вимог, що визначають її ефективність, зручність та безпеку використання. Насамперед, інтерфейс користувача має бути інтуїтивно зрозумілим і доступним навіть для недосвідчених користувачів, з урахуванням принципів UX/UI-дизайну. Система повинна мати адаптивне відображення для коректної роботи на різних типах пристроїв, включаючи настільні комп’ютери, планшети та смартфони. Особлива увага приділяється захисту персональних даних користувачів, що передбачає зберігання інформації в зашифрованому вигляді та використання захищених протоколів передавання даних (HTTPS).

Серед функціональних вимог — повноцінна підтримка всіх етапів онлайн-покупки: реєстрації, автентифікації, перегляду товарів, оформлення замовлень, надсилання повідомлень про статуси та взаємодії з кошиком. У контексті безпеки система має відповідати сучасним стандартам — передбачено реалізацію захищеного API-доступу з обов’язковою перевіркою токенів. Також передбачається розмежування прав доступу, що дозволяє обмежити можливості звичайних користувачів лише функціями перегляду та покупки, у той час як адміністратори мають змогу керувати вмістом каталогу, замовленнями та аналітикою.

Технічне завдання описане в Додатку А.

2.2. Вибір моделі розробки програмного продукту

Життєвий цикл розробки системи онлайн-торгівлі реалізовано у межах гнучкого підходу Agile з використанням Kanban як інструменту візуального управління процесами. Робота над проєктом поділяється на чіткі послідовні

фази, що ітеративно повторюються до завершення розробки повноцінного програмного продукту [14].

На початковому етапі проводиться збір функціональних та нефункціональних вимог до системи. Формується перелік ключових можливостей: реєстрація й автентифікація користувачів, перегляд товарів, робота з кошиком, оформлення замовлень, адміністрування контенту тощо. Уточнюються очікування кінцевих користувачів та особливості бізнес-логіки.

На основі вимог формується детальне технічне завдання, що містить опис функціоналу, структуру бази даних, умови інтеграції між клієнтською та серверною частинами, вимоги до продуктивності, безпеки та масштабованості. Цей документ виступає основою для планування задач у Kanban-дошці.

Архітектура базується на клієнт-серверній моделі. Визначається поділ на компоненти фронтенду (React-компоненти, маршрути, сторінки) та бекенду (роути, контролери, сервіси, модель даних MongoDB). Обираються ключові інструменти: React для інтерфейсу, Node.js з Express — для серверної частини, MongoDB — як гнучка NoSQL база даних, а також токени — для реалізації автентифікації та ролей доступу.

Кодування здійснюється поетапно відповідно до задач, виставлених у Kanban. Кожен функціональний блок створюється як окремий компонент React, що спрощує підтримку та тестування. Бекенд-логіка реалізується у вигляді REST API з чітким розділенням відповідальностей між маршрутами, контролерами та сервісами. Забезпечується обробка запитів, валідація даних і робота з базою.

Після реалізації функціональності відбувається її перевірка: проводиться ручне тестування основних сценаріїв взаємодії, перевіряється відповідність очікуваному результату, здійснюється тестування API та окремих компонентів. За потреби створюються автоматизовані тести для критичних ділянок коду.

Після завершення певного етапу розробки система проходить інтеграційне тестування, після чого деплоїться на сервер або хостинг.

На завершальному етапі система переходить в експлуатацію.

Здійснюється постійний моніторинг працездатності, збирання зворотного зв’язку від користувачів, аналіз помилок і розробка оновлень. Завдяки гнучкості Agile, нові функції можуть поступово впроваджуватись без зупинки всієї системи.

2.3. Архітектура та структура застосунку

У ході виконання кваліфікаційної роботи було розроблено вебзастосунок для онлайн-торгівлі, який реалізує повний цикл взаємодії користувача з системою — від перегляду товарів і додавання їх до кошика до оформлення замовлення. Система підтримує особисті кабінети користувачів, розмежування ролей (адміністратор/користувач), керування контентом та захищену автентифікацію. Проєкт побудовано із застосуванням сучасного стеку технологій — React для клієнтської частини, Node.js з Express на стороні сервера та MongoDB для зберігання даних.

Архітектура застосунку побудована на трирівневій моделі: клієнтська частина (інтерфейс користувача), серверна частина (логіка обробки запитів і взаємодія з базою даних) та база даних (MongoDB). Така структура забезпечує чітке розмежування відповідальностей між компонентами, спрощує масштабування системи, дозволяє ефективно реалізовувати нову функціональність та підвищує надійність підтримки застосунку. Крім того, це сприяє більшій гнучкості у процесі розробки, адже кожен рівень може розвиватися незалежно від інших.

2.3.1. Опис клієнтської частини проєкту

В структурі проєкту використаний підхід, рекомендований збирачем Vite для React-додатків. Було створено кореневий файл, конфігураційні файли, а також папки: для публікації коду, для збереження всіх завантажених пакетних модулів, для збереження публічних даних (зображень, іконок), для збереження компонентів (рис. 2.1).

Рисунок 2.1 – Представлення структури проєкту

Папка Dist містить готовий до випуску код, а саме: всі оптимізовані фото та іконки (зібрані в єдиний спрайт, що також в свою чергу оптимізований), кореневий файл index.html від якого відшовхується браузерний інтерпретатор, зібраний оптимізований JavaScript файл з усіма компонентами та використаними бібліотеками, зібраний та оптимізований файл стилів (CSS) з усіма модульними файлами стилізації компонент. Дана папка ігнорується в системі контролю версій, що визначає конфігураційний файл .gitignore.

Папка node_modules є ключовою частиною будь-якого проєкту, який використовує Node.js і керується системою керування пакетами npm (пакетний менеджер Node.js). Вона містить усі встановлені залежності, необхідні для роботи проєкту [19].

Папка public у вебпроєктах містить статичні файли, які повинні бути доступними для клієнта (браузера) без обробки або трансформацій. Ці файли копіюються без змін до вихідної директорії (dist) під час складання проєкту.

Папка src є основною директорією в проєкті, вона містить всі компоненти та файли стилізації. Весь вихідний код проєкту міститься в цій папці, після збирання він обробляється, компілюється та трансформується до відповідного формату, який використовує браузер.

У кореневій директорії розробленого вебзастосунку розміщено низку основних файлів, що забезпечують запуск, конфігурацію та супровід проєкту відповідно до стандартів сучасної веброзробки.

Файл index.html є головною HTML-сторінкою застосунку, з якої починається обробка клієнтського інтерфейсу браузером. Саме цей документ завантажується першим, після чого підключаються скрипти, стилі та інші ресурси, необхідні для рендерингу інтерфейсу користувача.

Файл .gitignore використовується для вказівки компонентів та файлів, які повинні бути виключені зі спостереження системи контролю версій Git. Це можуть бути тимчасові файли, каталоги зі збіркою, ключі доступу або залежності, що автоматично відновлюються, і не потребують збереження в репозиторії.

Файл package.json виконує роль основного конфігураційного документа проєкту. У ньому міститься інформація про назву, версію, опис, точку входу, а також список залежностей (бібліотек та модулів), які необхідні для функціонування застосунку. Також у цьому файлі визначаються скрипти для запуску сервера, збірки проєкту, тестування тощо.

Файл package-lock.json створюється автоматично під час встановлення залежностей і містить точні версії кожного пакета, що використовується в проєкті. Це гарантує стабільність збірки та однакове середовище незалежно від того, на якому комп’ютері або сервері запускається проєкт.

Основний код та організація компонентів, модулів, стилів та ресурсів необхідних для коректної роботи проєкту подані у папці src (рис 2.2).

Директорія pages містить файли, що відповідають за створення різних сторінок вебдодатку. У сучасних React-додатках ця структура допомагає організувати код та забезпечує зручне маршрутизацію між сторінками за

допомогою бібліотек, таких як React Router. Кожна сторінка є окремим компонентом, що виконує функцію основного контейнера для певного функціоналу чи розділу сайту. В даній директорії містяться компоненти каталогу та кошику, що представляють відповідні сторінки.

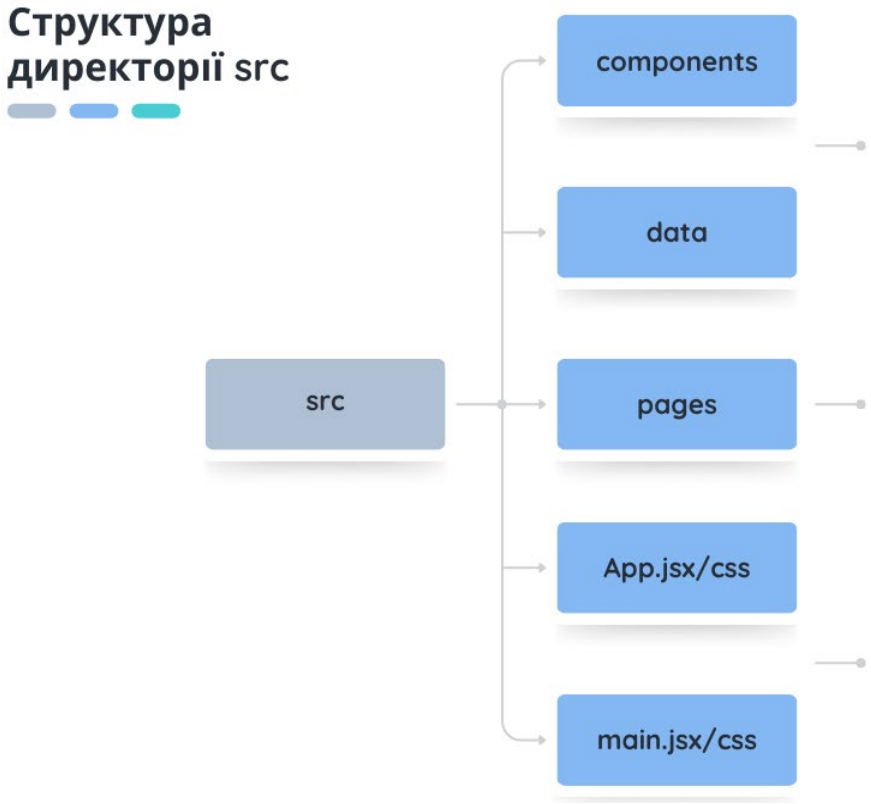

Рисунок 2.2 – Структура директорії src

Файл App.jsx є головним файлом React-додатка. Він відповідає за інтеграцію компонентів, сторінок та функціональності, забезпечуючи зв'язок між різними частинами додатка використовуючи модульність. У ньому налаштовується маршрутизація, підключаються глобальні стилі (App.css) та передається контекст додатка.

Файл main.jsx є точкою входу (коренем) додатку на основі React. Він відповідає за початкове завантаження React-компонентів у DOM (Document Object Model) та забезпечує інтеграцію з вебсторінкою.

Папка components (рис. 2.3) містить усі модулі статичного сайту, кожен модуль являє собою окремий елемент у загальній структурі, що може бути

перевикористаний в будь якому місці, що робить компонентний підхід зручним для сучасної розробки.

Кожна папка компонента містить файл з розширенням .jsx, що забезпечує розмітку та функціональність елемента на вебсторінці. Для кожного компонента створюється окремий файл стилів з розширенням .css або .scss, який визначає зовнішній вигляд елемента. Цей підхід дозволяє локалізувати стилі в межах компонента, уникаючи конфліктів з іншими елементами [16].

У разі складних компонентів, допоміжна логіка винесена в окремі файли, що сприяє більшій організованості та спрощує підтримку коду.

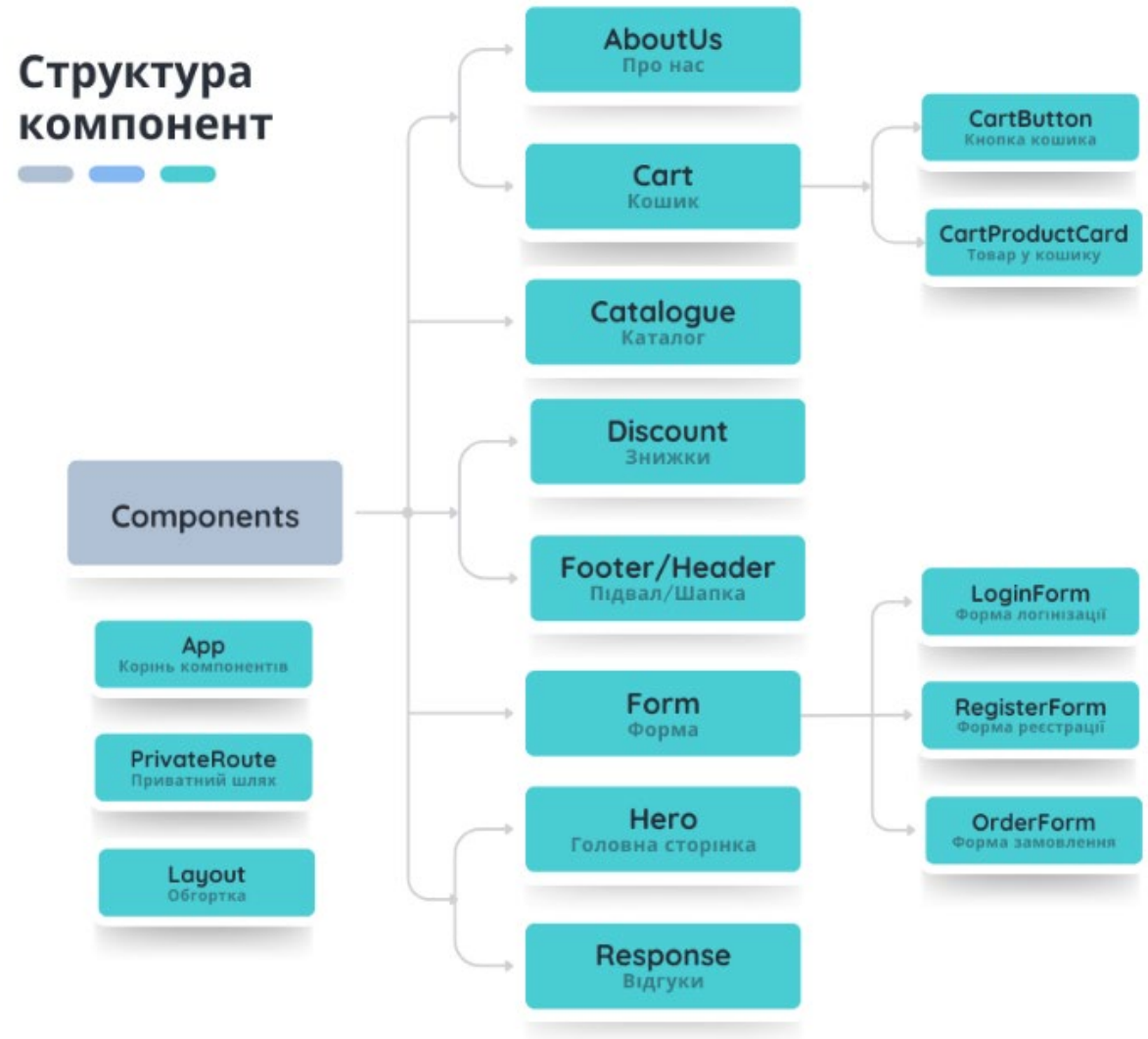

Рисунок 2.3 – Структура директорії components

AboutUs (Про нас) – компонент, що представляє блок інформації про магазин, а також відображає найпопулярніші стікери, з якими користувач може взаємодіяти.

Cart (Кошик) – декілька компонентів, що представляють собою окрему сторінку та модальне вікно, яке можна відкрити в будь-якому місці сайту.

Catalogue – компонент, що представляє окрему сторінку для показу всіх товарів, їх фільтрацію та пошук.

Discount – компонент, що відображає та забезпечує функціональність блоку, що показує користувачу товари зі знижками.

Header та Footer – компоненти, що представляє “шапку” та “підвал” вебсторінки відповідно, містять важливі посилання, контактну інформацію та елементи навігації по сторінці, що реалізована за допомогою бібліотеки “react-router-dom”. Дані елементи використовуються на різних сторінках, що зручно робити з допомогою компонентного підходу [6].

Form – компонент, що представляє форму для заповнення даних користувача, зокрема форму реєстрації, логінізації та замовлення.

Hero – перший блок, що зустрічає користувача на сайті, застосовується для зацікавленості користувача сайтом а в подальшому й продуктом.

Response – важливий компонент, що відображає відгуки клієнтів про продукт. Сприяє збільшенню довіри користувача до якості продукту.

Private/Restricted route – обгортки, що забороняються доступ неавторизованим/авторизованим користувачам.

Layout – загальна обгортка компонентів з початковими налаштуваннями сторінки.

2.3.2. Реалізація серверної частини вебдодатку

Серверна частина вебдодатку була реалізована з використанням JavaScript-середовища Node.js у поєднанні з фреймворком Express, що забезпечує зручну побудову RESTful API [15]. Основним завданням бекенду є обробка запитів клієнта, взаємодія з базою даних, автентифікація користувачів,

обробка замовлень та підтримка логіки функціонування онлайн-магазину. Структура серверного коду поділена на окремі модулі з чітким розмежуванням відповідальностей, що сприяє масштабованості та полегшує супровід проєкту (рис. 2.4).

Застосунок підтримує асинхронну обробку даних, що дозволяє ефективно обслуговувати одночасні запити без втрати продуктивності. У якості бази даних використовується MongoDB, що дозволяє зберігати інформацію у форматі JSON-документів, забезпечуючи гнучку роботу зі складними структурами даних. Впроваджено систему автентифікації користувачів із застосуванням токенів, що гарантує безпечний доступ до захищених маршрутів. Такий підхід дозволяє реалізувати сучасну, надійну та зручну серверну архітектуру для системи онлайн-торгівлі.

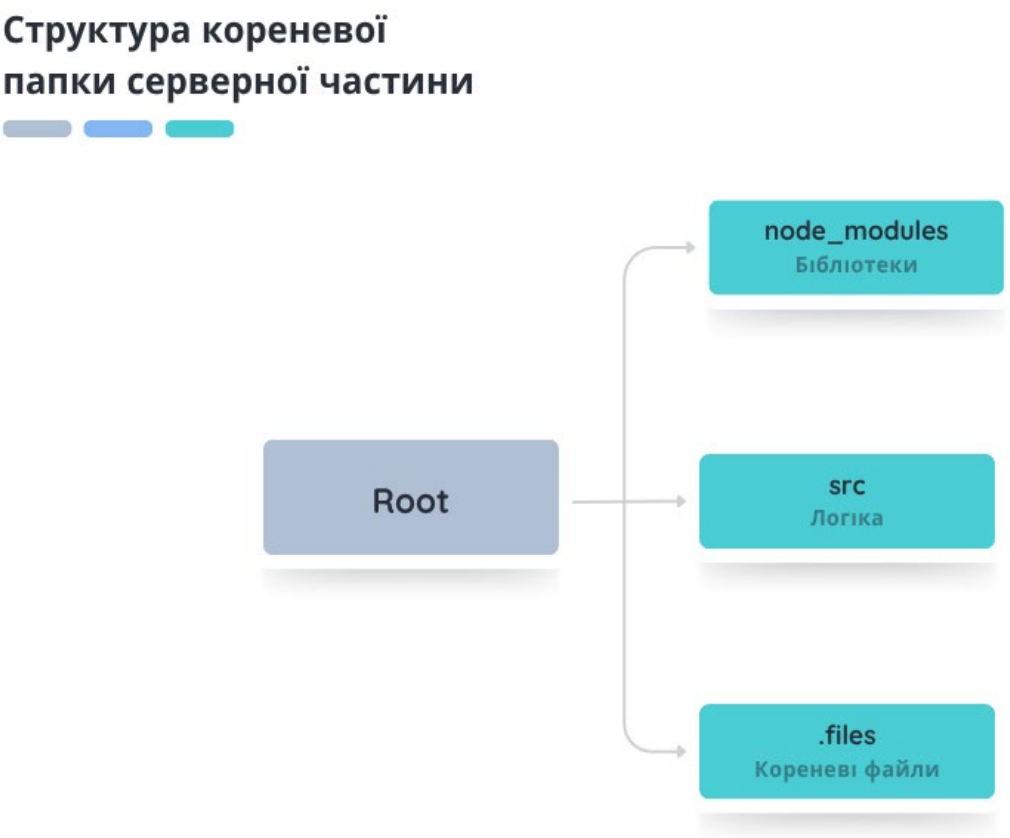

Рисунок 2.4 – Структура кореневої папки серверної частини

Структура кореневої папки серверної частини вебдодатку організована відповідно до принципів модульності та зручності в адмініструванні коду.

Такий підхід дозволяє швидко орієнтуватися в логіці проєкту, розширювати функціональність і підтримувати високий рівень читабельності. Нижче описано основні складові, що формують файлову структуру серверної частини.

node_modules – службова директорія, яка автоматично створюється після встановлення залежностей. Містить усі сторонні бібліотеки та пакети, що використовуються у проєкті.

src – основна робоча папка з вихідним кодом серверної логіки. Усередині розміщуються підкаталоги для моделей бази даних, маршрутів, контролерів, утиліт, middleware тощо (рис 2.5). Така структура сприяє розділенню логіки відповідно до призначення кожного модуля.

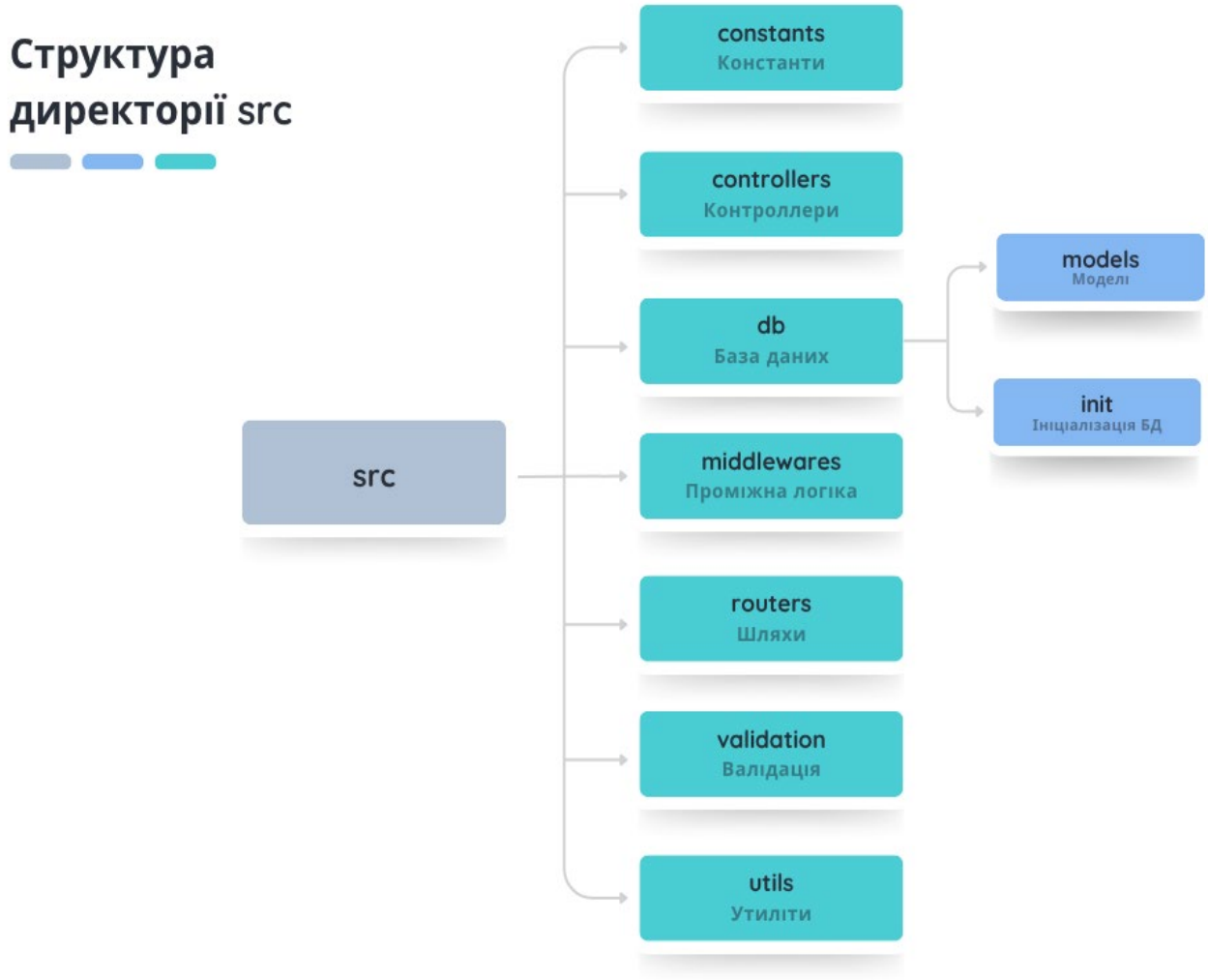

Рисунок 2.5 – Структура директорії src серверної частини

Кореневі файли – package.json описує основні характеристики проєкту, список залежностей та скрипти для запуску чи тестування застосунку. package-lock.json забезпечує фіксацію конкретних версій встановлених бібліотек. Файл .gitignore визначає, які каталоги й файли не повинні потрапляти до репозиторію. Головний файл запуску сервера index.js — відповідає за ініціалізацію застосунку, підключення бази даних, конфігурацію маршрутизації та запуск HTTP-сервера.

Директорія src у серверній частині вебдодатку є основною зоною зберігання бізнес-логіки та функціональних модулів. Вона організована за принципами модульності та розділення відповідальностей, що полегшує масштабування, налагодження й підтримку проєкту. Кожна папка в цій структурі виконує окрему роль у процесі обробки запитів і взаємодії з клієнтом та базою даних.

constants – містить файли з фіксованими значеннями, такими як статуси відповідей, повідомлення про помилки, конфігурації середовищ або типи ролей. Це забезпечує централізоване управління незмінними значеннями та сприяє уникненню дублювання коду.

controllers – включає функції-обробники запитів, які реалізують логіку взаємодії між маршрутизаторами та сервісами. Тут розміщуються функції для створення, читання, оновлення та видалення ресурсів (CRUD), які працюють із моделями бази даних.

db – містить підкаталог models, у якому визначено схеми MongoDB з використанням бібліотеки Mongoose, що описують структуру збережених документів. Файл init.js відповідає за підключення до бази даних, ініціалізацію моделей і обробку помилок підключення.

middlewares – включає проміжні функції, які обробляють запити до того, як вони потраплять до контролерів. Сюди можуть входити функції перевірки автентифікації, обробки помилок, логування запитів або обмеження доступу.

routers – реалізує маршрутизацію запитів на відповідні контролери згідно з REST-архітектурою. Кожен модуль маршрутизатора відповідає за певний ресурс і реєструє відповідні ендпоінти.

validation – містить схеми валідації запитів за допомогою Joi або іншої бібліотеки. Тут перевіряються вхідні дані на коректність перед передачею до контролерів.

utils – ключає допоміжні функції, які використовуються в різних частинах застосунку, наприклад, генерація токенів, хешування паролів, форматування дат чи обробку файлів.

2.3.3. Архітектура та реалізація бази даних

База даних є ключовим компонентом вебдодатку, оскільки забезпечує зберігання, доступ і обробку інформації, необхідної для функціонування системи. У рамках розробки проєкту було обрано документно-орієнтовану базу даних MongoDB, яка завдяки гнучкому підходу до структурування даних та високій масштабованості ідеально підходить для сучасних вебзастосунків. Її інтеграція здійснена за допомогою об'єктно-документного моделювання (ODM) через бібліотеку Mongoose, що дозволяє створювати чітко структуровані схеми даних, спростити запити до бази та забезпечити валідацію на рівні моделі [9].

У структурі бази даних центральне місце займає колекція користувачів (рис. 2.6), яка призначена для зберігання інформації про зареєстрованих користувачів системи. Вона містить такі ключові поля, як ім’я, електронна адреса, хешований пароль, роль користувача, а також додаткові технічні атрибути. Наявність цієї таблиці є критично важливою для забезпечення захищеного входу, управління сесіями та реалізації розмежування прав.

У базі даних також передбачено окрему колекцію session, яка відповідає за зберігання інформації про активні сесії користувачів. Вона містить такі поля, як ідентифікатор користувача, токен доступу, час створення та термін дії сесії. Ця таблиця відіграє важливу роль у забезпеченні безперервності авторизації,

відстеженні активності користувачів та безпечному управлінні життєвим циклом сесії. Структура колекції сесій відображена на рисунку 2.7.

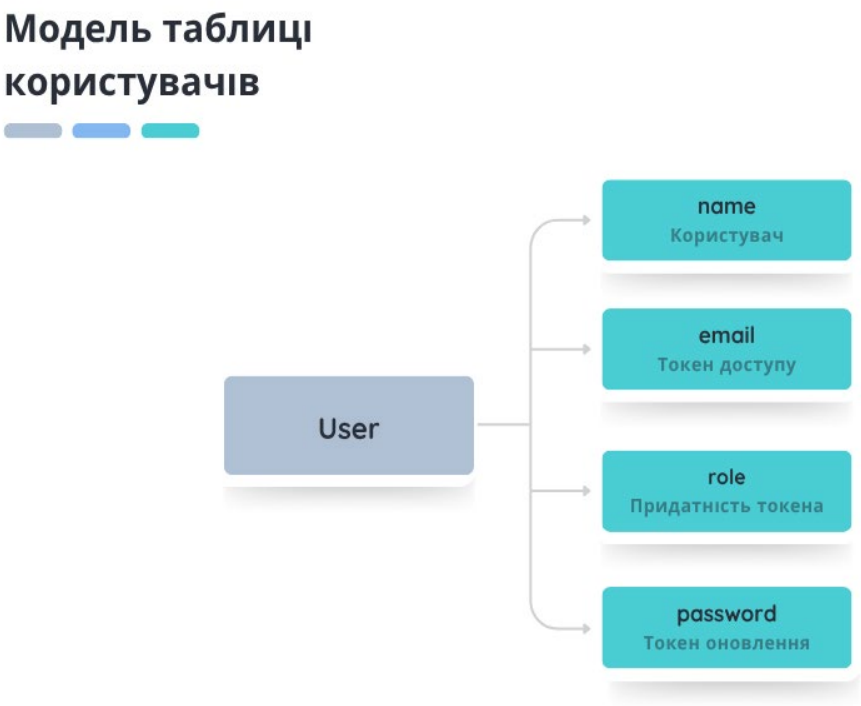

Рисунок 2.6 – Модель таблиці користувачів

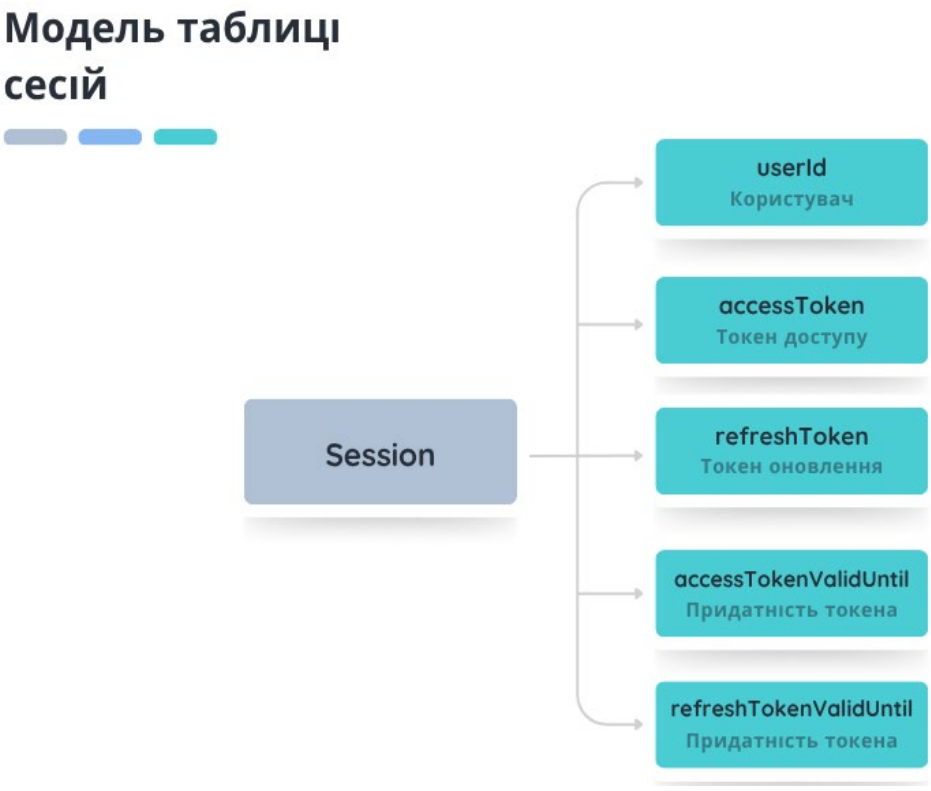

Рисунок 2.7 – Модель таблиці сесій

У структурі бази даних окрема колекція stickers відповідає за зберігання інформації про наклейки, що доступні для перегляду та купівлі в онлайн-магазині. До основних полів цієї таблиці належать: назва товару, опис, категорія, ціна, URL зображення, доступна кількість, а також інформація про те, чи додано товар до найпопулярніших (рис 2.8). Колекція наклейок є центральною в контексті організації каталогу продукції, оскільки саме вона забезпечує наповнення інтерфейсу магазину та підтримку основних операцій електронної комерції.

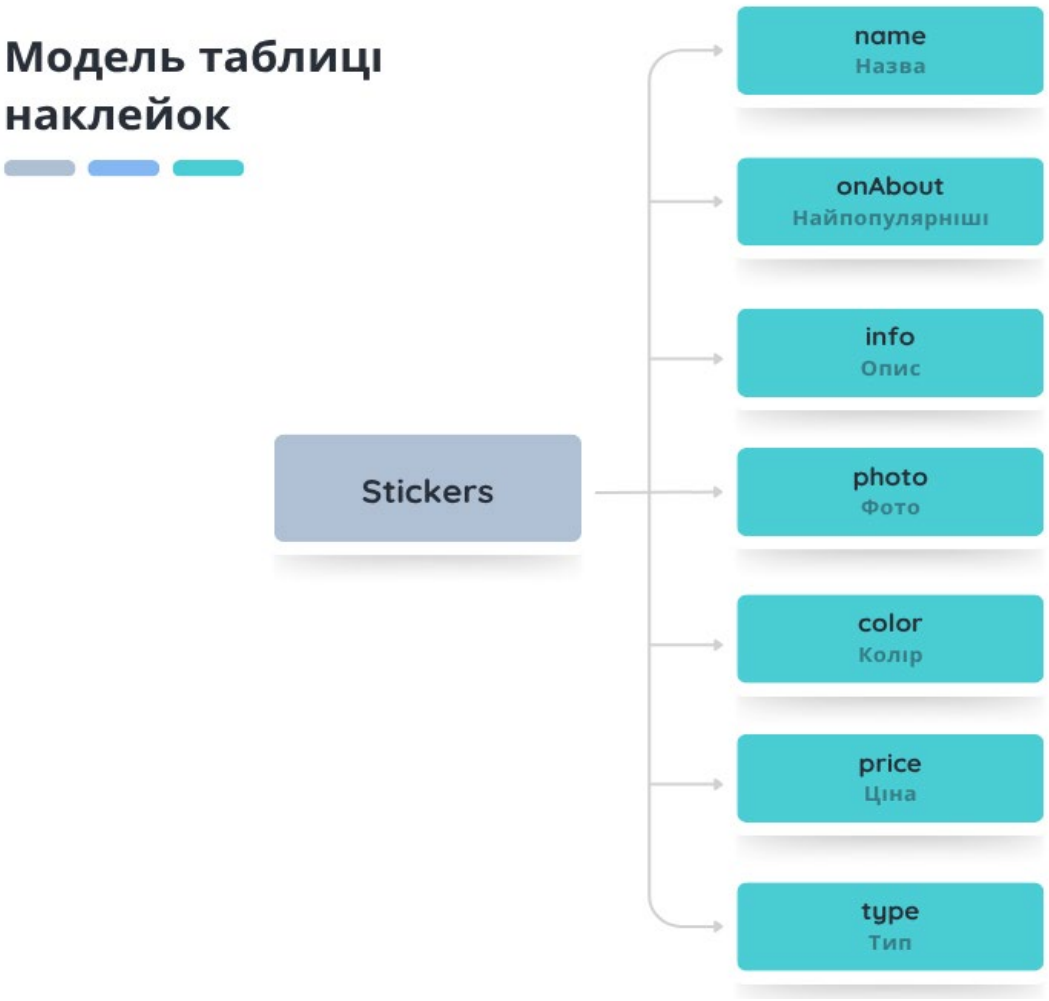

Рисунок 2.8 – Модель таблиці наклейок

Таким чином, архітектура та реалізація бази даних є ключовим елементом у забезпеченні функціонування клієнт-серверного застосунку. Чітко структуровані колекції, дозволяють ефективно зберігати та обробляти дані, що

стосуються користувачів, сеансів автентифікації та товарів відповідно. Ретельне проектування схеми бази даних забезпечує узгодженість, масштабованість і надійність під час взаємодії з іншими компонентами системи, що є основою для стабільної роботи онлайн-магазину в умовах реального навантаження.

2.3.4. Опис взаємодії загальної системи вебдодатку

Загальна взаємодія між компонентами клієнт-серверного вебдодатку є основою його коректного функціонування, оскільки забезпечує передачу, обробку та збереження інформації у відповідь на дії користувача. У ході реалізації проєкту застосовано класичну модель взаємодії, за якої ініціатором запиту виступає клієнтська частина, що за допомогою HTTP-запитів звертається до серверного API [11]. Сервер, обробивши запит, здійснює відповідну логіку — автентифікацію, перевірку прав доступу, валідацію вхідних даних — і звертається до бази даних для зчитування або модифікації інформації.

У результаті отримані дані перетворюються у відповідний формат і повертаються клієнту для подальшого відображення. Така організація забезпечує чітке розмежування відповідальностей між компонентами, дозволяє масштабувати систему, підвищує надійність і забезпечує розширюваність при майбутньому доповненню функціональності [17].

На діаграмі (рис. 2.9) послідовно представлено ланцюг взаємодій: користувач ініціює дію на інтерфейсі клієнтської частини, яка надсилає HTTP-запит до сервера. Серверна частина, у свою чергу, обробляє запит за допомогою відповідного контролера, проводить перевірку даних через middleware та звертається до бази даних MongoDB для отримання або зміни інформації. Після обробки запиту результат передається назад до клієнта, де відбувається оновлення інтерфейсу відповідно до відповіді сервера.

Узагальнюючи опис взаємодії клієнтської частини, серверної логіки та бази даних, можна зробити висновок, що чітка організація процесів обміну даними є запорукою стабільної та ефективної роботи вебдодатку. Завдяки

компонентному підходу на фронтенді, модульній структурі серверної частини та використанню гнучкої бази даних MongoDB, система забезпечує швидке реагування на дії користувача, належну обробку запитів і коректне збереження даних. Така взаємодія сприяє підтримці масштабованості, спрощує процес тестування та супроводу, а також створює надійне підґрунтя для подальшого розширення функціональності застосунку.

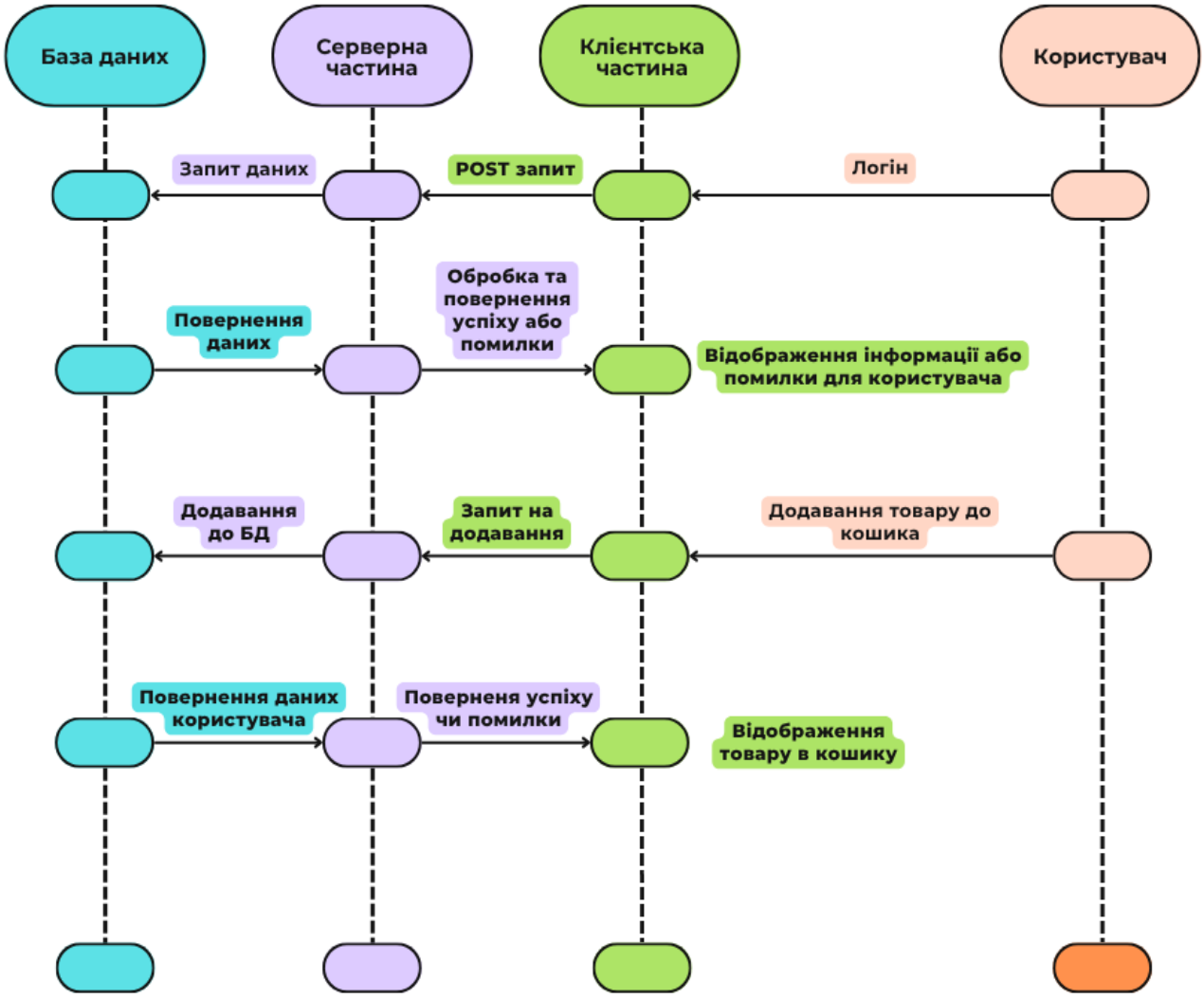

Рисунок 2.9 – UML-діаграма взаємодії компонент додатку

2.4. Обґрунтування вибору технологій та інструментів для системи

У процесі розробки вебдодатку надзвичайно важливим етапом є обґрунтований вибір технологій та інструментів, які забезпечать ефективне

функціонування системи, її масштабованість, безпеку та зручність у підтримці. Враховуючи вимоги до проєкту, обсяг очікуваних даних, а також потребу в сучасному, динамічному інтерфейсі, було обрано набір технологій, які оптимально поєднуються між собою та відповідають поточним стандартам у сфері веброзробки.

2.4.1. Обґрунтування вибору технологій розробки клієнтської частини

Для реалізації клієнтської частини було обрано бібліотеку React, яка є однією з найпопулярніших технологій сучасної фронтенд-розробки. Основною перевагою React є її компонентний підхід, що дозволяє створювати багаторазово використовувані та ізольовані елементи інтерфейсу. Це значно спрощує підтримку та масштабування коду. Завдяки віртуальному DOM, React забезпечує високу продуктивність навіть у складних інтерфейсах з великою кількістю оновлень.

Для обміну даними з сервером було обрано бібліотеку Axios, яка підтримує обробку HTTP-запитів у форматі Promise. Вона забезпечує просту конфігурацію, обробку помилок, інтерцептори запитів/відповідей та зручну інтеграцію з токенами авторизації, що робить її ідеальним вибором для RESTful-інтерфейсів.

Як збирач проєкту було обрано Vite — сучасний інструмент для швидкої розробки фронтенду, який забезпечує миттєве оновлення модулів без перезавантаження сторінки. Завдяки використанню нативних ES-модулів та попередній компіляції залежностей, Vite значно пришвидшує запуск і збірку проєкту, що є особливо важливим під час активної фази розробки [2].

Для навігації всередині застосунку використовується бібліотека React Router, яка дозволяє реалізувати маршрутизацію на стороні клієнта. Це дає змогу створювати SPA (single-page application), де перемикання між сторінками відбувається без повного перезавантаження, покращуючи користувацький досвід і швидкість роботи інтерфейсу.

Для управління глобальним станом застосунку було використано бібліотеку Redux. Вона забезпечує централізоване зберігання даних, що дозволяє ефективно синхронізувати інформацію між різними компонентами. Такий підхід особливо корисний у складних інтерфейсах, де багато елементів взаємодіють з однаковими даними. Використання Redux сприяє підвищенню масштабованості застосунку та спрощує його підтримку завдяки передбачуваному оновленню стану через систему дій (actions) та редукторів (reducers) [6].

Для роботи з формами було обрано бібліотеку Formik, яка значно спрощує обробку вводу користувача, управління станом форми та її валідацією. Formik дозволяє створювати масштабовані та зручні у використанні форми з мінімальними витратами на написання додаткового коду. Вона інтегрується з іншими бібліотеками та підтримує перевірку полів у режимі реального часу, що покращує користувацький досвід [13].

Разом із Formik для реалізації валідації форм використовується бібліотека Yup. Вона дозволяє визначати схеми перевірки даних у декларативному стилі, забезпечуючи зручний синтаксис і високу гнучкість у створенні правил. Yup допомагає уникнути помилок вводу користувача, гарантує відповідність даних заданим вимогам і покращує загальну безпеку застосунку.

Для швидкого створення стильного та адаптивного інтерфейсу застосунку використано бібліотеку компонентів MUI, що реалізує принципи дизайну від Google — Material Design. MUI надає велику кількість готових елементів управління (кнопки, форми, модальні вікна, таблиці тощо), які легко налаштовуються та інтегруються з React. Це дозволяє зосередитися на логіці додатку, не витрачаючи надмірний час на розробку інтерфейсу з нуля.

2.4.2. Обґрунтування вибору технологій розробки серверної частини

Для реалізації серверної частини застосунку обрано середовище виконання JavaScript — Node.js. Воно дозволяє створювати високопродуктивні, масштабовані серверні рішення з використанням єдиної мови програмування як

на клієнті, так і на сервері. Асинхронна модель обробки запитів надає можливість ефективно керувати великою кількістю з'єднань одночасно, що є критично важливим для сучасних вебзастосунків [7].

На базі Node.js використано мінімалістичний фреймворк Express.js, який спрощує створення серверних маршрутів, обробку запитів і відповідей, а також підключення проміжних обробників (middleware). Express надає гнучку структуру для організації логіки взаємодії з клієнтом і базою даних, що дозволяє зберігати розробку простою, підтримуваною та розширюваною [8].

Для зберігання даних застосовується документно-орієнтована база даних MongoDB. Її гнучка структура дозволяє працювати з неструктурованими або напівструктурованими даними, а також легко адаптуватися до змін у схемі зберігання. Завдяки високій продуктивності та масштабованості, MongoDB ідеально підходить для динамічних вебпроєктів.

Для зручної роботи з MongoDB використано бібліотеку Mongoose, яка забезпечує об'єктно-документне моделювання (ODM). Вона дозволяє створювати схеми моделей, здійснювати перевірку даних на рівні моделі, а також значно спрощує взаємодію із запитами до бази [12]. Mongoose сприяє підтримці чистої архітектури серверної частини та підвищує безпеку даних.

Для валідації вхідних даних на рівні маршрутизаторів застосовується бібліотека Joi. Вона дозволяє створювати гнучкі та надійні схеми перевірки запитів, що надходять від клієнта, ще до обробки серверною логікою. Це забезпечує додатковий рівень захисту від некоректних або шкідливих даних та зменшує ризики помилок на етапі збереження інформації в базу даних.

2.5. Особливості програмної реалізації системи онлайн-торгівлі

Реалізація системи онлайн-торгівлі передбачає врахування низки технічних особливостей, спрямованих на забезпечення стабільної, масштабованої та зручної у користуванні платформи. Особливу увагу приділено архітектурному поділу проєкту, обробці даних у реальному часі,

підтримці безперервної інтерактивності, а також захисту персональних даних користувачів.

Серед ключових аспектів — впровадження системи авторизації, модульне управління товарами та користувачами, використання REST API для взаємодії клієнта з сервером, обробка помилок та забезпечення відповідності бізнес-логіки вимогам вебкомерції. Такий підхід гарантує ефективну роботу додатку в умовах змінних навантажень і високих вимог до якості сервісу.

Автентифікація та авторизація користувачів. У вебдодатку реалізовано механізми автентифікації (рис 2.10) та авторизації користувачів для забезпечення безпеки доступу до персоналізованих функцій системи. Автентифікація здійснюється через перевірку облікових даних під час входу, після чого користувач отримує токен доступу, який використовується для підтвердження особи при подальших запитах. Авторизація визначає рівень доступу до ресурсів: наприклад, звичайний користувач може переглядати товари та керувати кошиком, тоді як адміністратор має додаткові можливості з управління вмістом каталогу. Такий підхід дозволяє ефективно розмежовувати права доступу, захищати дані та підтримувати безпечну логіку взаємодії із системою.

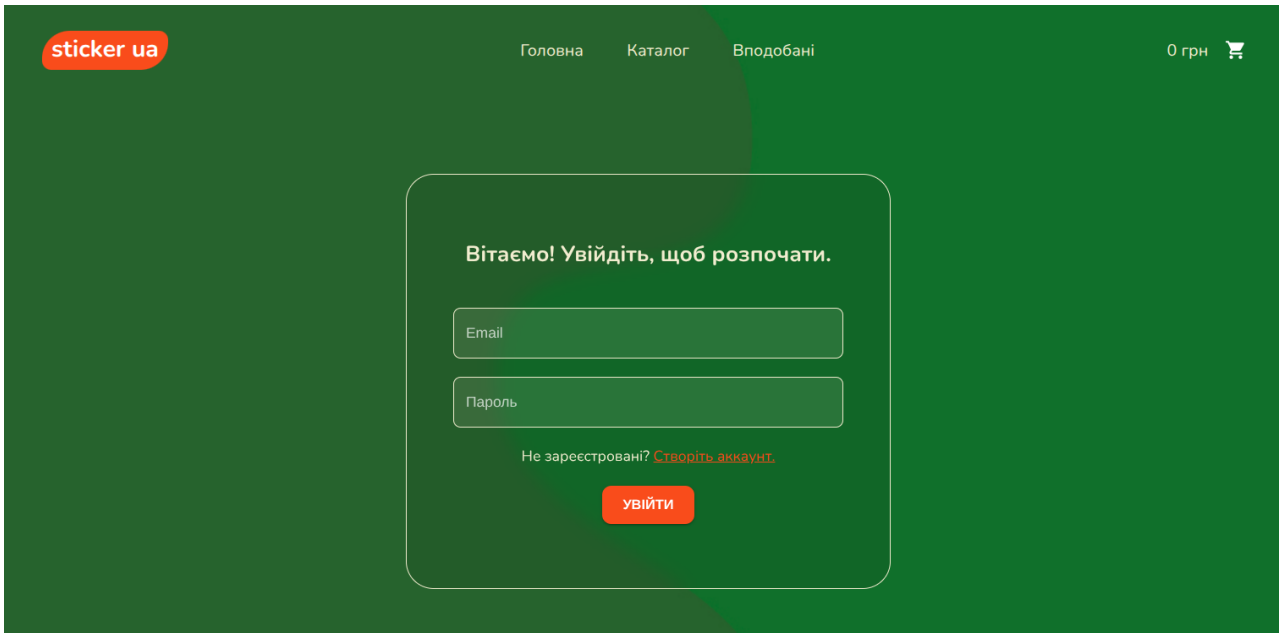

Рисунок 2.10 – Форма входу користувачів

Управління користувацьким інтерфейсом. Управління користувацьким інтерфейсом є одним із ключових аспектів створення зручного та інтуїтивно зрозумілого вебдодатку. У розробці інтерфейсу було враховано сучасні підходи до побудови UI-компонентів, що дозволяють забезпечити логічну структуру сторінок, швидку реакцію на дії користувача та адаптивність до різних розмірів екранів. Основна увага зосереджена на взаємодії з каталогом товарів, роботою з кошиком та процесом оформлення замовлення.

Каталог товарів (рис 2.11) реалізовано у вигляді сітки стікерів, що відображаються за допомогою компонентів React. Кожна картка товару містить зображення, назву, ціну та кнопку для додавання в кошик. Інтерфейс дозволяє користувачеві легко переглядати доступні стікери, застосовувати фільтри та шукати потрібні позиції за ключовими словами. Завдяки реактивності інтерфейсу зміни у фільтрації чи пошуку відбуваються миттєво.

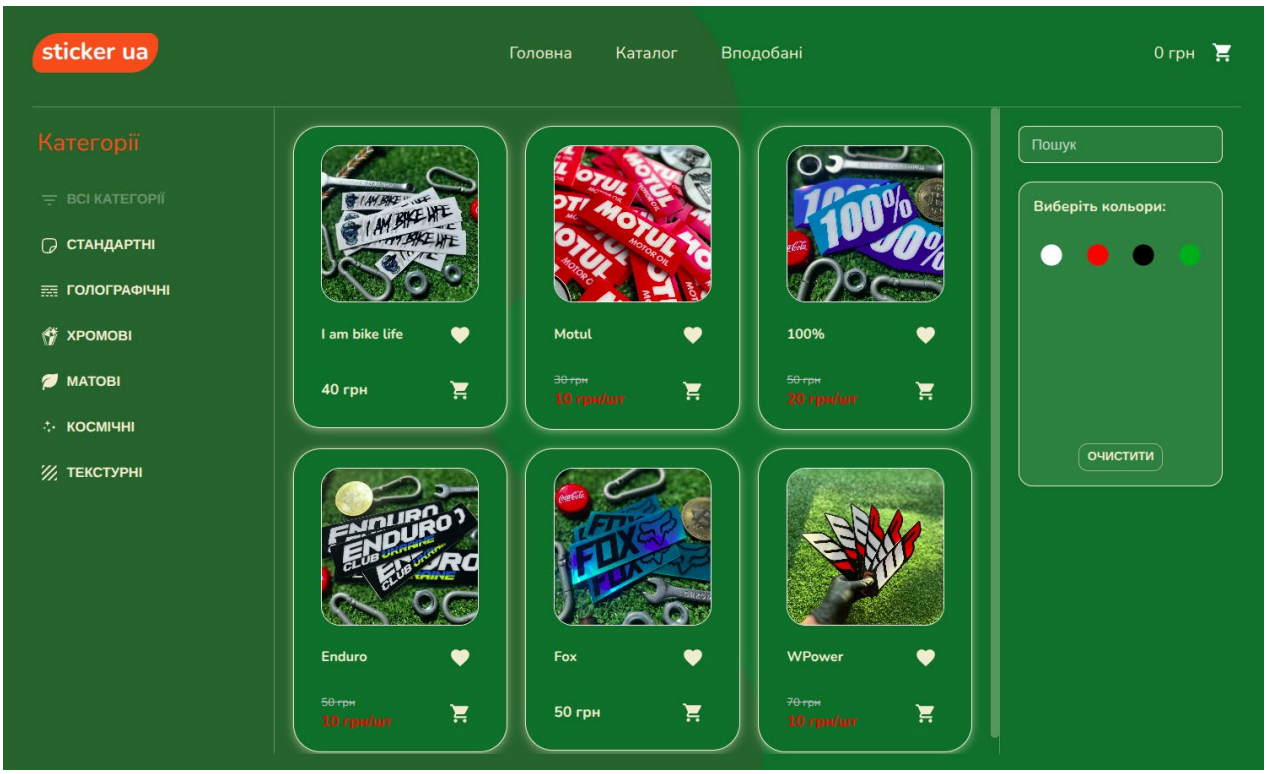

Рисунок 2.11 – Каталог товарів

Додавання товарів у кошик реалізовано шляхом взаємодії з глобальним станом застосунку (через Redux). При натисканні кнопки «Додати в кошик»

відповідний товар додається до списку обраних, і користувач може переглядати вміст кошика у будь-який момент (рис 2.12). У кошику відображаються всі додані позиції, кількість кожного товару, загальна сума та можливість редагування або видалення окремих елементів.

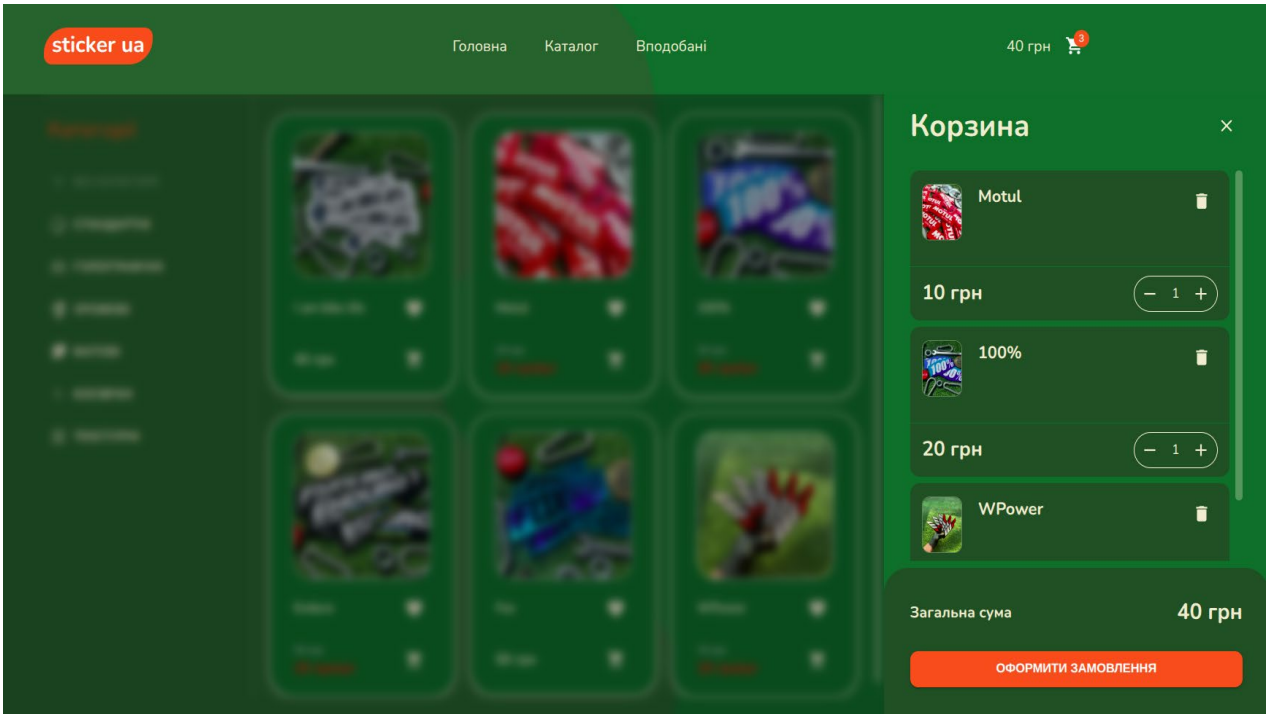

Рисунок 2.12 – Кошик з продуктами

Оформлення замовлення реалізовано у вигляді покрокової форми, де користувач вводить необхідні контактні та дані доставки (рис 2.13). Для перевірки правильності введення інформації застосовується бібліотека Yup у зв’язці з Formik. Після успішного заповнення форми дані передаються на сервер для обробки, а користувач отримує повідомлення про підтвердження замовлення. Такий підхід забезпечує зрозумілий та безперебійний процес купівлі, підвищуючи загальну зручність користування системою.

Окрему роль у вебдодатку відіграє панель адміністратора, яка надає можливість керування вмістом каталогу. Адміністратор має змогу додавати нові стікери (рис. 2.14), редагувати існуючі позиції або видаляти їх у разі потреби. Для цього реалізовано зручну форму з полями для введення назви, опису, ціни, категорії та завантаження зображення товару. Усі зміни миттєво

відображаються в інтерфейсі користувача, що забезпечує актуальність і повноту інформації на сайті. Такий функціонал сприяє ефективному управлінню асортиментом та підтримці високої якості обслуговування клієнтів.

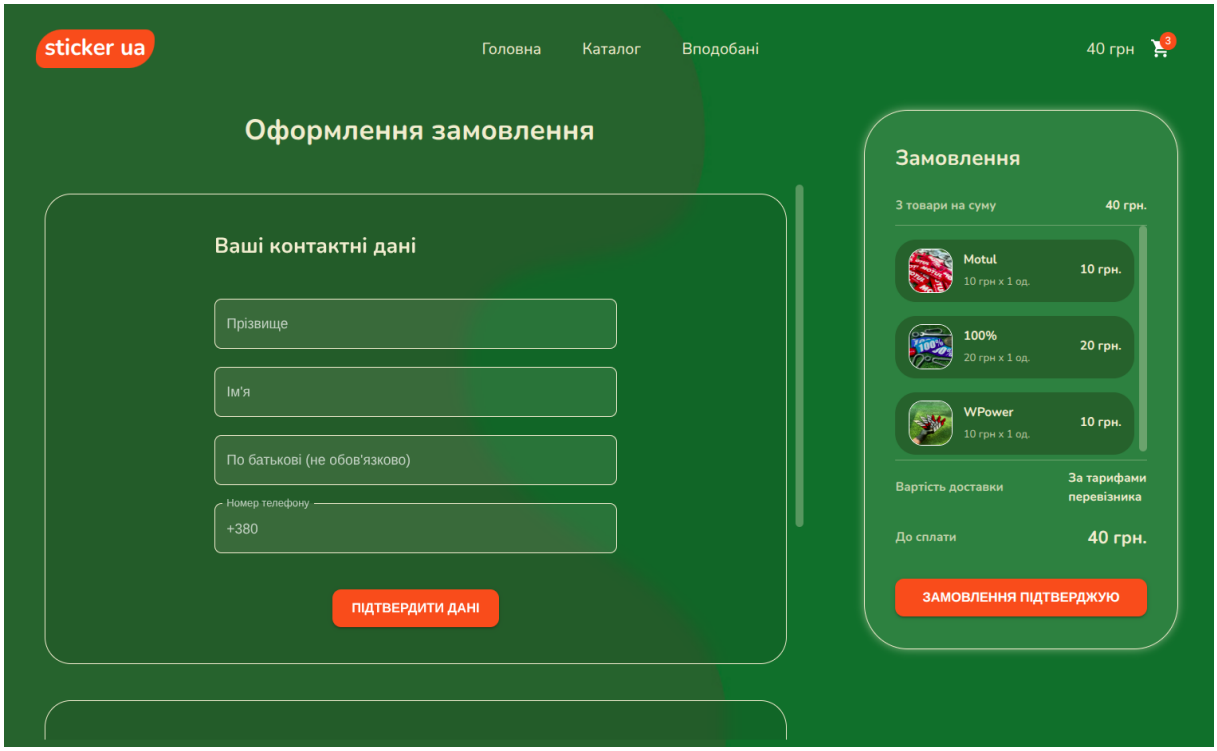

Рисунок 2.13 – Оформлення замовлення

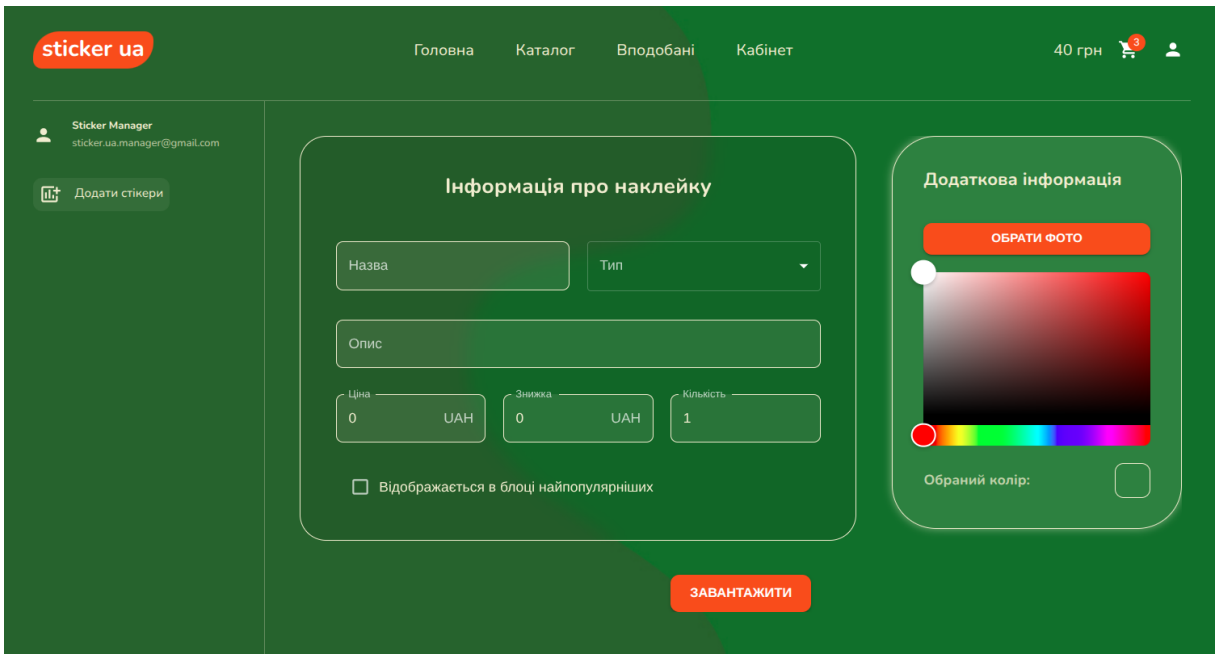

Рисунок 2.14 – Додавання інформації наклейки в БД

Загалом, реалізований користувацький інтерфейс поєднує сучасні технології та підходи, що забезпечують зручність, швидкодію та адаптивність вебдодатку. Чітка структура компонентів, інтеграція з глобальним станом та використання бібліотек для валідації форм дозволяють користувачеві без зайвих зусиль взаємодіяти з каталогом товарів, керувати вмістом кошика та оформлювати замовлення. Такий підхід сприяє позитивному користувацькому досвіду й підвищує ефективність онлайн-продажів.

Керівництво користувачу наведено в Додатку Б

2.6. Організація тестування та налагодження програмного засобу

Успішна реалізація вебдодатку передбачає не лише розробку інтерфейсу та функціоналу, а й ґрунтовне тестування та налагодження, що забезпечують стабільність і надійність програмного засобу. На цьому етапі особливу увагу було приділено перевірці коректної роботи як клієнтської, так і серверної частин, а також взаємодії між ними. Тестування проводилося з метою виявлення помилок, логічних недоліків, порушень у роботі інтерфейсу та обміні даними з базою.

У межах клієнтської частини тестуванню підлягали основні функціональні модулі: каталог товарів, кошик, форма оформлення замовлення та адміністративна панель. Застосовувалися як ручні, так і автоматизовані методи перевірки поведінки інтерфейсу в різних умовах — зокрема зміна фільтрів, пошук товарів, додавання до кошика, редагування та видалення позицій. Важливою складовою стала перевірка правильності роботи глобального стану (Redux) та валідації форм за допомогою бібліотек Formik і Yup. Це дало змогу запобігти відправці неповних або некоректних даних.

Для ефективного налагодження інтерфейсу активно використовувалися інструменти React Developer Tools, Redux DevTools та засоби розробника в браузері. За їх допомогою здійснювався моніторинг стану компонентів, потоку даних та швидке виявлення помилок у логіці чи візуальному представленні.

Паралельно проводилося тестування серверної частини, яка забезпечує роботу з базою даних та обробку запитів користувачів. Було перевірено стабільність роботи REST API: коректне виконання CRUD-операцій, обробка замовлень, додавання та редагування товарів, автентифікація користувачів. Для цього застосовувалися інструменти Postman і ручне тестування. Також перевірялася обробка помилок на сервері та коректність відповідей у разі некоректного запиту чи недопустимого доступу до захищених маршрутів.

У результаті комплексного підходу до тестування і налагодження було досягнуто високого рівня стабільності та надійності роботи вебдодатку. Усі виявлені недоліки були систематизовані й виправлені до моменту розгортання проєкту, що забезпечило якісну взаємодію користувачів із системою та безперебійне функціонування програмного забезпечення в умовах реального використання.

2.7. Рекомендації щодо використання та впровадження програмного продукту

Для ефективного використання розробленого вебдодатку необхідно дотримуватись певних технічних та організаційних умов впровадження. Оскільки система складається з клієнтської та серверної частин, важливо правильно розгорнути обидва компоненти, забезпечивши стабільну роботу сервісу та доступність бази даних.

Серверну частину доцільно розгортати на хмарному сервері або VPS з попередньо встановленим Node.js та менеджером пакетів (npm або yarn). Необхідно налаштувати середовище виконання (зокрема, змінні середовища з параметрами підключення до бази даних, портами, тощо) та забезпечити захищений канал обміну даними (HTTPS). Для зберігання інформації про товари, замовлення та користувачів використовується база даних MongoDB, яка може бути розгорнута локально або на хмарному сервісі (наприклад, MongoDB

Atlas). Слід подбати про налаштування прав доступу до колекцій, створення резервних копій та моніторинг продуктивності.

Клієнтську частину рекомендовано розміщувати на хостингу з підтримкою статичних файлів (наприклад Vercel), попередньо зібравши застосунок за допомогою Vite. Для коректної роботи необхідно забезпечити правильну маршрутизацію та доступ до API сервера з відповідного домену, а також налаштувати CORS.

Отже, для повноцінного впровадження програмного продукту потрібно забезпечити коректне налаштування серверного середовища, бази даних та клієнтської частини. Дотримання вищезазначених рекомендацій дозволить гарантувати стабільну роботу системи, безперебійний доступ користувачів до функціоналу та подальше масштабування за потреби.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було здійснено повний цикл розробки клієнт-серверної вебсистеми онлайн-торгівлі з використанням односторінкової архітектури (SPA). Проведене дослідження підтвердило актуальність теми, що зумовлена стрімким розвитком SPA архітектури в сфері електронної комерції. Систему реалізовано з урахуванням сучасних вимог до швидкодії, адаптивності та зручності користування.

В межах дослідження здійснено аналіз особливостей функціонування клієнт-серверних систем у сфері онлайн-торгівлі, зокрема вивчено сучасні підходи до побудови SPA-застосунків, їх переваги над традиційною багатосторінковою архітектурою та можливості масштабування. У результаті було визначено функціональні та нефункціональні вимоги до системи, що дозволило спроєктувати архітектуру системи.

Реалізовано інтерфейс користувача за принципом компонентно-орієнтованого підходу. Було впроваджено визначений функціонал для вебсистеми онлайн-торгівлі. Інтерфейс реалізовано з урахуванням сучасних вимог до зручності та адаптивності, що забезпечує інтерактивну та ефективну взаємодію з користувачем.

Серверна частина побудована на Node.js із застосуванням REST API, що забезпечує ефективну обробку запитів, а також взаємодію з базою даних MongoDB. Реалізовано CRUD-операції для керування товарами та обробки замовлень, а також механізми аутентифікації й авторизації. Також було розроблено й налагоджено систему взаємодії між клієнтською та серверною частиною, що дозволяє виконувати операції в режимі реального часу без повного оновлення сторінки.

Було здійснено тестування працездатності програмного продукту, зокрема функціональне тестування основних модулів та перевірка стабільності роботи під час типових сценаріїв використання, а також виконано оцінку

продуктивності та можливостей масштабування. Використання бібліотек валідації дозволило контролювати коректність введених користувачем даних.

Сформовано рекомендації щодо розгортання системи в реальному середовищі, включаючи серверну інфраструктуру, підключення до хмарної бази даних MongoDB та хостинг клієнтської частини. Враховано питання безпеки, зокрема налаштування CORS, захищені API-запити, обмеження прав доступу та зберігання конфіденційної інформації. Такий підхід забезпечує можливість стабільної роботи системи та її масштабування в майбутньому.

Проєкт має потенціал для подальшого розвитку. Зокрема, можливе підключення платіжних систем для реалізації повноцінного процесу онлайн-оплати, удосконалення особистого кабінету користувача, впровадження розширених звітів для адміністратора, багатомовної підтримки інтерфейсу та системи сповіщень. Подібні доповнення дозволять перетворити систему на повноцінну платформу електронної комерції.

Отже, розроблена клієнт-серверна вебсистема продемонструвала свою ефективність як сучасне рішення для організації онлайн-продажів. Вона забезпечує зручну взаємодію з користувачем, швидку обробку запитів, надійне управління даними та має масштабовану архітектуру, що дозволяє адаптувати систему до зростаючих потреб бізнесу. Завдяки використанню сучасних технологій як на клієнтській, так і на серверній стороні, система є готовою до подальшого розширення функціоналу та інтеграції нових можливостей.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *React. React documentation.* URL: https://react.dev/
2. *Vite. Vite Guide.* URL: https://vite.dev/
3. Смирнов І. Розробка односторінкових додатків. *WebCase*. 16.02.2021 URL: https://webcase.com.ua/uk/blog/razrabotka-odnostranichnyh-prilozhenij-spa-webcase/
4. Crockford D. Javascript: the good parts : навч. посіб. O'Reilly Media, Incorporated, 2008.
5. Web Components - Web APIs | MDN. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_components
6. Для чого і коли використовується redux. FoxMinded. URL: https://foxminded.ua/shcho-take-redux/
7. *Node.js*. *NodeJs documentation*. URL: https://nodejs.org/docs/latest/api/
8. *Express*. *Express guide*. URL: https://expressjs.com/
9. *MongoDB*. *MongoDB documentation*. URL: https://www.mongodb.com/docs/
10. *QATestLab*. *Токени аутентифікації*. 06.07.2023. URL: https://training.qatestlab.com/blog/technical-articles/authentication-tokens/
11. Клієнт-серверна архітектура. *Wikipedia*. 29.01.2023. URL: https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D1%96%D1%94%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0
12. Jamie M. Вступ до Mongoose для MongoDB та Node.js. *EnvatoTuts+*. URL: https://code.tutsplus.com/uk/an-introduction-to-mongoose-for-mongodb-and-nodejs--cms-29527a
13. Overview. *Formik Docs*. URL: https://formik.org/docs/overview
14. Канбан. *Brain rain*. URL: https://brainrain.com.ua/uk/shcho-take-kanban/
15. Структура проектів на Node.js. *DevZone*. URL:

https://devzone.org.ua/post/node-hero-chastyna-7-struktura-proektiv-na-nodejs

16. Кучер Т. Модульна архітектура для розширеного React-застосунку. *High Bar Journal*. 20.09.2022. URL: https://journal.gen.tech/post/modules-for-react-app
17. Клієнт-серверна архітектура. *QATestLab*. 28.05.2020. URL: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/
18. REST API: що це, як працює, переваги і приклади. *GoIT*. URL: https://goit.global/ua/articles/rest-api-shcho-tse-iak-pratsiuie-perevahy-i-pryklady/
19. Southall T. React component with npm using Vite. Tom Southall. URL:https://tomsouthall.com/blog/publishing-react-component-using-vite
20. Структура сайту або інформаційна архітектура. Створення сайтів, розробка програмного забезпечення. URL: https://coi.ua/blog/Cbc/Website-Structures-How-to-Choose-the-Best-Option-for-Your-Web-Project/

ДОДАТКИ

Додаток А.

Технічне завдання

Вступ

Програмний продукт, що розробляється, має назву “STICKER UA”. Він призначений для підтримки процесів електронної комерції, зокрема для надання користувачам можливості переглядати каталог товарів, формувати замовлення, додавати обрані товари до кошика чи списку бажань, а також здійснювати базову взаємодію з особистим кабінетом. Окрім базових функцій для покупців, програмний продукт також передбачає адміністративну панель, що дозволяє авторизованим адміністраторам додавати, редагувати або видаляти товари з каталогу, а також переглядати замовлення та взаємодіяти з базою даних через зручний інтерфейс. Система призначена для використання в малому онлайн-бізнесі, зокрема інтернет-магазинах цифрових чи фізичних товарів, і може бути розгорнута на хостинг-платформі з підтримкою Node.js та MongoDB.

Підстави для розробки

Розробка програмного продукту “STICKER UA” здійснюється на основі технічного завдання, затвердженого під час розробки та складання розгорнутого плану дослідження. Найменування теми розробки – “Розробка клієнт-серверної системи онлайн-торгівлі з підходом односторінкової архітектури”

Призначення розробки

Функціональне призначення програми полягає у створенні зручного вебінструменту для вибору, перегляду та замовлення наклейок через інтернет. Програмний продукт орієнтований на кінцевих користувачів, які можуть взаємодіяти з системою через інтуїтивно зрозумілий інтерфейс, реалізований у форматі односторінкової вебзастосунку (SPA). Основні функції системи включають можливість переглядати каталог товарів, згрупованих за

категоріями, ознайомлюватися з характеристиками наклейок, їх зображеннями та описами, а також додавати обрані товари до кошика або списку бажань.

З точки зору експлуатації, система орієнтована як на кінцевих споживачів, так і на власників малого та середнього бізнесу, які можуть використовувати її для просування своїх товарів. Менеджерські функції включають можливість керування каталогом через адміністративний інтерфейс — додавання, редагування або видалення товарів. Передбачена також інтеграція з логістичними службами для автоматичного створення електронних накладних. Усе це дозволяє зробити процес продажу наклейок більш ефективним, зменшуючи потребу в ручній обробці замовлень та підвищуючи рівень обслуговування клієнтів.

Вимоги до програмного продукту

Вимоги до функціональних характеристик. Програмний продукт передбачає основні можливості системи, необхідні для забезпечення її коректного функціонування, зручності користування та задоволення потреб цільової аудиторії. Згідно з поставленими завданнями, функціонал клієнтської частини повинен включати механізми реєстрації та аутентифікації користувачів, можливість перегляду товарного каталогу з фільтрацією за категоріями, додавання товарів до кошика та списку бажаного, перегляд детальної інформації про товари (зображення, характеристики, опис), а також оформлення замовлення з вибором способу доставки та оплати.

Крім того, система має реалізовувати функції керування обліковим записом, зокрема перегляд історії замовлень, редагування особистої інформації та керування списком бажань. Для адміністративної частини передбачено інтерфейс керування товарною номенклатурою: додавання, редагування та видалення товарів, завантаження зображень, встановлення актуальних цін та характеристик. Також передбачається реалізація функціоналу формування та друку електронних накладних для логістичних служб. Система повинна забезпечувати інтерактивність, динамічне оновлення вмісту без повного

перезавантаження сторінки та високу швидкодію при виконанні запитів, що відповідає концепції односторінкової архітектури.

Умови експлуатації. Користування системою передбачається в середовищі сучасних веббраузерів на настільних та мобільних пристроях, що підтримують технології HTML5, CSS3 та JavaScript. Клієнтська частина реалізована за допомогою фреймворку React, що забезпечує інтерактивний інтерфейс, а також адаптивність відображення сторінок для різних розмірів екранів. Система не вимагає встановлення на пристрій користувача, оскільки доступ до функціоналу надається через інтернет. Для забезпечення стабільної роботи рекомендується підключення до мережі з мінімальною швидкістю 10 Мбіт/с.

Серверна частина системи функціонує на платформі Node.js з використанням бази даних MongoDB, яка забезпечує збереження, обробку та обслуговування користувацьких і товарних даних. Система потребує розгортання на хостинговому середовищі з підтримкою Node.js (версії не нижче 18) та доступом до бази даних MongoDB. Також необхідно забезпечити наявність SSL-сертифіката для безпечного обміну даними та відповідність вимогам безпеки (зокрема захист паролів через хешування, валідацію даних і захист від типових атак). Продукт розрахований на постійну експлуатацію без потреби в спеціалізованому технічному обслуговуванні, за умови забезпечення працездатності серверної інфраструктури.

Вимоги до складу і параметрів технічних засобів. Для клієнтської частини системи достатньо сучасного пристрою з доступом до мережі Інтернет. Зокрема, користувач повинен мати персональний комп’ютер, ноутбук або мобільний пристрій з роздільною здатністю дисплея щонайменше 1280×720 пікселів, оперативною пам’яттю від 2 ГБ та встановленим веббраузером, що підтримує сучасні вебтехнології (HTML5, CSS3, JavaScript ES6+). Рекомендовано використовувати актуальні версії браузерів Google Chrome, Mozilla Firefox, Microsoft Edge або Safari. Також передбачається наявність стабільного інтернет-з’єднання.

Серверна частина застосунку повинна бути розгорнута на хостинговій або фізичній машині з відповідними обчислювальними ресурсами. До мінімальних вимог належать процесор класу не нижче 4-ядерного Intel i5 або аналогічного AMD, обсяг оперативної пам’яті від 8 ГБ (рекомендовано – 16 ГБ), твердотільний накопичувач обсягом від 50 ГБ, а також встановлене середовище виконання Node.js (версія 18 або новіша). Сервер повинен мати можливість забезпечити надійне виконання серверної логіки, маршрутизацію запитів, обробку замовлень та взаємодію з базою даних.

База даних реалізується на основі MongoDB, яка є документоорієнтованою СУБД та дозволяє ефективно зберігати й обробляти дані про товари, замовлення, користувачів та інші ресурси системи. Вона може бути розміщена як локально, так і у хмарному середовищі (наприклад, через MongoDB Atlas). Для стабільної роботи бази даних достатньо конфігурації з 2 віртуальними ядрами, 4 ГБ оперативної пам’яті та стабільного з’єднання із сервером застосунку. Обов’язковою є наявність механізмів резервного копіювання та моніторингу стану бази для запобігання втраті інформації.

Вимоги до транспортування і збереження. Передача даних між клієнтським інтерфейсом та сервером повинна здійснюватися із використанням захищених протоколів (зокрема HTTPS), що гарантує конфіденційність та цілісність інформації під час її транспортування мережею. Для захисту від атак типу слід використовувати SSL-сертифікати та додаткові механізми шифрування, включаючи авторизацію за токенами. Передбачено застосування сучасних засобів валідації даних як на стороні клієнта, так і на стороні сервера для мінімізації ризиків зловмисного втручання.

Щодо збереження, дані користувачів, інформація про замовлення, товари та адміністративні дії повинні зберігатися на сервері з дотриманням політик резервного копіювання та захисту від втрати даних. Серверна частина системи повинна бути розгорнута на хмарному середовищі або виділеному сервері з обмеженим доступом, використовуючи системи контролю доступу, шифрування паролів (bcrypt), а також регулярні оновлення безпеки. База

даних (MongoDB) повинна бути налаштована з використанням ролей доступу, журналювання змін та резервного копіювання з періодичністю, що відповідає політиці безпеки підприємства.

Вимоги до програмної документації. Документація для клієнт-серверної вебсистеми повинна включати повний комплект технічних та користувацьких матеріалів, необхідних для її впровадження, експлуатації, обслуговування та подальшого розвитку. Уся документація має бути створена відповідно до чинних стандартів розробки програмного забезпечення з урахуванням вимог до структурованості, логічної послідовності викладу та доступності для різних цільових груп: розробників, адміністраторів та кінцевих користувачів.

До обов’язкових компонентів документації належать:

1. Технічне завдання, у якому визначено загальні характеристики продукту, функціональні вимоги, архітектурні рішення та обмеження щодо середовища використання.

2. Розробницька документація, що містить опис архітектури вебсистеми, структури бази даних, логіки клієнтської та серверної частин, інтерфейсів API, а також інструкції щодо локального розгортання і супроводу продукту.

3. Користувацька інструкція, яка пояснює, як працювати з інтерфейсом вебсайту: перегляд каталогу, додавання товарів до кошика або списку бажань, оформлення замовлень та управління обліковим записом.

4. Документація з тестування, що охоплює тест-плани, тест-кейси, звіти про результати функціонального та нефункціонального тестування, включаючи перевірку надійності, безпеки та продуктивності.

5. Рекомендації з інформаційної безпеки, що містять принципи захисту персональних даних користувачів, рекомендації з використання HTTPS, захисту від атак, а також процедури резервного копіювання і відновлення.

За потреби може бути передбачена розробка додаткової документації, зокрема щодо локалізації інтерфейсу, адаптації під мобільні пристрої чи

відповідності специфічним вимогам замовника або міжнародним стандартам. Усі документи мають бути підтримувані в актуальному стані та зберігатися у доступному форматі в системі контролю версій або в онлайн-репозиторії.

Техніко-економічні показники

Орієнтовна економічна ефективність розробки вебзастосунку полягає у підвищенні прибутковості онлайн-продажів завдяки автоматизації процесу оформлення замовлень, зменшенню витрат на обслуговування клієнтів та оптимізації управління товарним асортиментом. Система дозволяє скоротити час обробки замовлень, зменшити кількість помилок під час комунікації з клієнтами, а також підвищити рівень задоволеності користувачів завдяки зручному та інтуїтивно зрозумілому інтерфейсу.

Економічні переваги розробки полягають у її гнучкості, можливості масштабування під конкретні бізнес-завдання, а також у нижчих витратах на підтримку в порівнянні з універсальними комерційними платформами. Наявність інструментів інтеграції з системами доставки та аналітики забезпечує додаткову ефективність у веденні бізнесу.

Стадії і етапи розробки

Першою є стадія ***аналізу вимог***, в межах якої здійснюється збір інформації щодо потреб користувачів та функціональних очікувань від системи. Проводиться аналіз конкурентних рішень, визначаються ключові функції системи (каталог товарів, кошик, облікові записи, оформлення замовлень). На основі зібраної інформації формується технічне завдання, визначається структура даних, обирається технологічний стек і створюється первинна архітектура.

Друга стадія — ***проєктування***, яка охоплює створення архітектурної моделі вебінтерфейсу, клієнтської частини на React, серверної частини на Node.js або Spring Boot, бази даних PostgreSQL та налаштування REST API для інтеграції. Виконується моделювання структури бази даних, схеми маршрутизації запитів, логіки взаємодії між модулями системи, а також формується технічна документація.

Третій етап — ***розробка***, на якому реалізується функціональність клієнтської та серверної частини. Frontend створюється на базі React із використанням бібліотек для маршрутизації, форм і управління станом. Backend реалізовується з використанням Node.js або Spring Boot, здійснюється налаштування серверів, база даних інтегрується з бізнес-логікою. Також підключається система обліку користувачів, додавання до кошика, оформлення замовлень.

Четверта стадія — ***тестування та налагодження***. Вона охоплює функціональне тестування, перевірку безпеки (авторизація, доступ до даних), навантажувальне тестування, перевірку API та валідацію візуального інтерфейсу. Виправляються помилки, оптимізується продуктивність.

П'ятим етапом є ***впровадження***, у межах якого здійснюється розгортання вебзастосунку на хостингу або сервері, генерація фінальної документації для користувачів і адміністраторів.

Останнім, постійним етапом є ***технічна підтримка та обслуговування***. Він включає усунення помилок, випуск оновлень, внесення змін відповідно до нових вимог або технічних змін у сторонніх сервісах. Цей етап триває на безперервній основі та виконується командою технічної підтримки та розробниками. На всіх етапах передбачається узгодження технічної документації, протоколів змін, інструкцій і звітів про виконані роботи.

Порядок контролю і приймання

Для забезпечення якості та стабільної роботи вебзастосунку передбачено проведення комплексного контролю у вигляді декількох видів тестування. Основну увагу буде приділено функціональному тестуванню, яке дозволить перевірити ключові функції системи: реєстрацію та авторизацію користувачів, перегляд каталогу товарів, роботу кошика, оформлення замовлень та адміністративну панель управління.

Тестування безпеки охоплюватиме перевірку механізмів захисту персональних даних користувачів, авторизації та контролю доступу до ресурсів системи. Особлива увага буде приділена шифруванню паролів, захисту API та

уникненню типових вебзагроз.

Також буде проведено навантажувальне тестування, щоб оцінити, як система реагує на великий потік одночасних користувачів — з метою перевірки швидкості завантаження сторінок, обробки запитів до бази даних та ефективності кешування.

На етапі інтеграційного тестування перевірятиметься узгоджена робота клієнтської частини, серверного застосунку, бази даних та сторонніх сервісів (наприклад, API платіжної системи або системи доставки). Це дозволить переконатися в цілісності та взаємодії усіх компонентів платформи.

Після успішного завершення усіх видів тестування та виправлення виявлених помилок розпочинається процес приймання. Приймання програмного продукту “STICKER UA” здійснюється за відповідністю вимогам технічного завдання. Якщо вебзастосунок працює стабільно, не містить критичних помилок та виконує всі передбачені функції, проєкт вважається завершеним.

Додаток Б.

Інструкція користувачу

Загальні відомості

Система онлайн-продажів «STICKER UA» створена для продажу тематичних наклейок для двоколісного транспорту. Автор програми — Сюх Олександр. Програмний продукт складається з клієнтської частини, розробленої на React.js, та серверної частини на Node.js з базою даних MongoDB. Сайт доступний як на комп’ютерах, так і на мобільних пристроях. Інструкція описує основні можливості застосунку, вимоги до середовища та взаємодію із системою.

Функціональне призначення

Система дозволяє користувачу реєструватися, переглядати каталог товарів, додавати товари в кошик або до обраного, оформлювати замовлення та переглядати історію покупок. Для адміністратора передбачено можливості додавання, редагування та видалення товарів, а також керування замовленнями.

Умови застосування програми

Сайт «STICKER UA» доступний для використання через будь-який сучасний браузер без потреби встановлення додаткового програмного забезпечення. Інтернет-з'єднання бажане для перегляду товарів, додавання їх у кошик та оформлення замовлення. У разі тимчасової втрати з'єднання дані про вибрані товари можуть зберігатися локально в браузері, а синхронізація відбудеться після відновлення доступу до мережі.

Повідомлення користувачу

При некоректному введенні даних або проблемах із сервером користувач може побачити такі повідомлення: «Невірний email або пароль», «Сервер недоступний. Спробуйте пізніше» або «Цей email вже використовується». У таких випадках рекомендується перевірити правильність введених даних або наявність підключення до Інтернету.

Реєстрація та вхід

Користувач має змогу створити обліковий запис, вказавши email і пароль. Авторизовані користувачі можуть переглядати замовлення, додавати товари до обраного та створювати замовлення.

Перегляд товарів

Каталог товарів поділений на категорії, що дає змогу фільтрувати продукти. Кожна позиція містить зображення, опис, ціну та кнопки для додавання в кошик чи до списку бажань. Передбачено адаптивне відображення для мобільних пристроїв. У кошику можна переглянути додані товари, змінити їх кількість або видалити. Для оформлення замовлення потрібно заповнити форму з контактними даними. Після підтвердження замовлення користувач отримає сповіщення про його успішну обробку.

Адміністративна панель

Кабінет доступний лише користувачам із правами адміністратора. Вона дозволяє додавати нові товари, редагувати існуючі, видаляти позиції, а також переглядати замовлення користувачів.

Рекомендації користувачу

Для стабільної роботи сайту рекомендується використовувати сучасні браузери (Google Chrome, Firefox, Edge) з увімкненим JavaScript. У разі виникнення технічних проблем слід очистити кеш браузера або звернутись до адміністратора через форму зворотного зв’язку. Дотримуючись цієї інструкції, користувач зможе безперешкодно взаємодіяти із системою, здійснювати покупки, керувати своїм акаунтом та отримувати якісне обслуговування.

АНОТАЦІЯ

Сюх О. М. – Розробка клієнт-серверної системи онлайн-торгівлі з підходом односторінкової архітектури. Рукопис.

Кваліфікаційна робота за спеціальністю 122 Комп'ютерні науки. – Волинський національний університет імені Лесі Українки, Луцьк. – 2025 р.

Робота присвячена створенню вебсистеми, яка дозволяє здійснювати онлайн-продаж товарів, зокрема наліпок, у багатокористувацькому середовищі. Система реалізована як односторінковий застосунок (SPA), що забезпечує швидку взаємодію з інтерфейсом без повного перезавантаження сторінки. Вона включає клієнтську частину, розроблену з використанням React, Vite, JavaScript, та серверну логіку з REST API.

Під час виконання роботи було розглянуто різні варіанти побудови архітектури подібних систем, визначено ролі користувачів (адміністратор, покупець), способи фільтрації та відображення товарів, реалізовано модулі каталогу, кошика, списку бажань і оформлення замовлень. Інформація про товари зберігається у JSON-форматі з можливістю підключення до бази даних MongoDB. Також реалізовано адаптивну верстку та зручну навігацію.

Мета роботи – створити зручний, масштабований і сучасний вебзастосунок для електронної комерції з інтуїтивним інтерфейсом, що дозволяє ефективно взаємодіяти з користувачем і виконувати базові функції онлайн-магазину.

У результаті створено повноцінну систему, яка складається з клієнтської частини та REST API, підтримує інтерактивну роботу користувача, динамічне відображення даних і готова до подальшого розширення функціоналу, зокрема інтеграції з платіжними сервісами.

Ключові слова: вебзастосунок, SPA, онлайн-торгівля, React, JavaScript, Vite, REST API, MongoDB, клієнт-серверна архітектура, електронна комерція.