МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ПІДРУЦЬКИЙ ДМИТРО АНАТОЛІЙОВИЧ

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ О. В. Зелінська
«____»_____ 20__ р.

ВЕБСАЙТ ІНТЕРНЕТ-МАГАЗИНУ ВІЙСЬКОВОГО СПОРЯДЖЕННЯ

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Т. В. Січко, доцент кафедри
інформаційних технологій,
к. т. н., доцент

_____

Оцінка: _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____

Вінниця – 2025

АНОТАЦІЯ

Підруцький Д.А Розробка вебсайту інтернет-магазину військового спорядження. Спеціальність 122 «Комп'ютерні науки», освітня програма «Комп'ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2025.

У кваліфікаційній (бакалаврській) роботі досліджено та проаналізовано поняття вебдодатку, виділені основні їх принципи та створення. За допомогою таких технологій як: HTML, CSS, JavaScript, бібліотеки React, Node.js був розроблений вебдодаток, основна мета якого – продаж товарів військового спорядження.

Ключові слова: вебдодаток, веброзробка, інтернет-магазин, JavaScript, React, Node.js.

74 ст., 39 рис., 2 дод., 40 джерел.

ABSTRACT

Pidrutskyi D.A. Development of a web application for a military equipment online store. Specialty 122 «Computer Science», educational program «Computer Science». Vasyl's Stus Donetsk National University, Vinnytsia 2025.

In the qualification (bachelor's) work, the concept of a web application is researched and analyzed, its main principles and development process are outlined. Using technologies such as HTML, CSS, JavaScript, the React library, and Node.js, a web application was developed with the primary purpose of selling military equipment products.

Keywords: web application, web development, online store, JavaScript, React, Node.js.

74 pages, 39 figures, 2 appendices, 40 references.

ЗМІСТ

ВСТУП

З огляду на сучасні реалії, де дедалі більше людей здійснюють покупки онлайн, розробка інтернет-магазину військового спорядження стає особливо актуальною. Інтернет надає клієнтам змогу швидко та зручно купувати необхідні товари незалежно від їхнього місцезнаходження чи часу доби. Для продавців це відкриває можливість розширити аудиторію, оптимізувати процеси продажу та збільшити прибуток. Військове спорядження належить до специфічної категорії товарів, яка вимагає особливої уваги до питань безпеки, юридичних обмежень і зручності вибору для кінцевого користувача.

Створення вебсайту є складним і багатокомпонентним процесом, що вимагає знань у сфері програмування, дизайну, маркетингу та кібербезпеки. Важливо не лише забезпечити коректну роботу інтернет-магазину, а й створити зручний інтерфейс користувача, налаштувати систему онлайн-оплат і подбати про захист особистих даних клієнтів.

У дипломній роботі розглядається процес розробки вебсайту інтернет-магазину, який спеціалізується на продажі військового спорядження. Це особливо актуально, оскільки така продукція потребує зручної категоризації, фільтрації та чіткої інформації щодо характеристик товарів. Крім того, потрібно враховувати вимоги до реалізації окремих категорій спорядження та забезпечити швидке оформлення замовлень.

Мета роботи полягає у створенні сучасного, функціонального та безпечного вебсайту для інтернет-магазину, що спеціалізується на реалізації військового спорядження, з урахуванням потреб цільової аудиторії, специфіки продукції та її стандартів.

Об’єктом дослідження є процес розробки вебресурсів для комерційних організацій.

Предметом дослідження є методи, інструменти та технології вебпрограмування, які застосовуються для створення інтернет-магазину військової тематики, а також реалізація його функцій, захисту даних

користувачів і забезпечення зручної взаємодії з вебресурсом.

Для досягнення мети потрібно реалізувати такі завдання:

- створити структуру бази даних для зберігання інформації про товари, користувачів, замовлення, кошик і транзакції;
- розробити інтерфейс для перегляду товарів, їхніх характеристик, додавання до кошика та обробки замовлень;
- реалізувати систему реєстрації та входу з хешуванням паролів за допомогою bcryptjs;
- впровадити авторизацію через JWT-токени з контролем доступу для користувачів і адміністраторів;
- створити адміністративну панель для керування товарами, перегляду та редагування замовлень;
- реалізувати пошукову систему з підтримкою часткових збігів, фільтрацією за категоріями та сортуванням за ціною;
- інтегрувати платіжну систему Stripe для безпечної онлайн-оплати;
- забезпечити динамічне оновлення кошика з обрахунком загальної вартості товарів;
- впровадити механізми перевірки дійсності токена та автоматичного виходу при його завершенні;
- провести тестування всіх модулів і перевірити безперебійність роботи ключових функцій.

Апробація результатів дослідження. Результати кваліфікаційної (бакалаврської) роботи апробовано на:

- XI Міжнародній науково-практичній конференції студентів, аспірантів і молодих учених «Актуальні проблеми гуманітарних, технічних і природничих наук», яка відбулась 30 квітня 2025 року на базі ДонНУ імені Василя Стуса. Тези на тему: «Розробка e-commerce сайту для продажу військового спорядження» опубліковано у збірнику матеріалів конференції.
- VI Всеукраїнській науково-практичній конференції здобувачів вищої

освіти та молодих вчених «Прикладні інформаційні технології», яка відбулась 22 травня 2025 року в online-форматі. Тези на тему: «Розробка вебсайту для продажу військового спорядження» опубліковано у збірнику матеріалів конференції.

- V Всеукраїнської науково-практичної конференції здобувачів, аспірантів та молодих вчених «Прикладні інформаційні технології», яка відбулась 24 травня 2024 року в online-форматі. Тези на тему: «Розробка погодного вебдодатка з використанням visual crossing weather API» були опубліковані в електронному збірнику наукових праць, який можна переглянути за посиланням https://jait.donnu.edu.ua.

Отримані результати корисні для тих, хто планує створювати власний сайт військової продукції або інший онлайн-магазин з товарами, які вимагають особливих умов продажу. Вони допомагають краще зрозуміти основні етапи створення такого сайту, вибір технологій і підходів до розробки, а також ефективні методи організації роботи сайту.

Структура дипломної роботи включає теоретичний аналіз принципів створення вебсайтів електронної комерції, огляд технологій, які використовують для створення сайту, та опис ключових етапів процесу розробки. У фінальній частині підбиваються підсумки щодо ефективності обраних рішень і їхньої практичної реалізації.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ВЕБСАЙТУ

1.1. Огляд існуючих розробок

Сьогодні вебсайти — це вже не розкіш, а необхідність. Ми постійно стикаємось із ними: читаємо новини, замовляємо їжу, шукаємо інформацію, спілкуємось, купуємо речі. Саме тому, важливим є не просто вміти робити сайти, а розуміти, які вони бувають, навіщо потрібні і як їх найкраще реалізовувати з технічної точки зору.

Загалом сайт — це набір сторінок, які мають спільну тему та об'єднані під одним доменом. Вони розміщуються на сервері, щоб користувачі могли заходити туди з будь-якого пристрою [1].

Вебсайти бувають різними — все залежить від того, яку задачу вони мають виконувати. Найпоширеніші типи [2]:

- інформаційні сайти;
- сайти-візитки;
- блоги;
- портфоліо;
- форуми;
- корпоративні сайти;
- освітні платформи;
- соціальні мережі;
- інтернет-магазини.

Інформаційні сайти мають на меті донесення корисної інформації до користувачів. Вони часто складаються з кількох розділів: головна сторінка, новини, статті, контакти тощо. Головна функція такого сайту — це надання актуальної інформації в зручному вигляді.

Вимоги:

- чітка і зрозуміла навігація;
- зручна система пошуку;

- можливість оновлення контенту;
- мобільна адаптація;
- швидкість завантаження сторінок.

Сайти-візитки – це прості сайти, які здебільшого складаються з кількох сторінок: «Про нас», «Послуги», «Контакти». Вони служать для того, щоб представити компанію або особу в Інтернеті.

Вимоги:

- лаконічний дизайн;
- швидкість завантаження;
- простота в управлінні;
- інтеграція з контактними формами або чатами;
- оптимізація для мобільних пристроїв;

Блоги дозволяють користувачам публікувати статті, постійно оновлювати контент та взаємодіяти з аудиторією через коментарі, лайки тощо. Блог може мати категорії, теги, пошукову систему для зручного знаходження статей.

Вимоги:

- система управління контентом;
- функціональність коментарів і рейтингу;
- інтеграція з соціальними мережами;
- гнучкість у публікаціях, а саме можливість додавати текст, зображення, відео.

Портфоліо використовується для демонстрації робіт, проектів, досягнень. Це може бути сайт дизайнера, фотографа, програміста або художника, який хоче показати найкращі роботи.

Вимоги:

- галерея зображень або відео;
- фільтри для сортування робіт, наприклад, за типом проекту;
- зворотний зв'язок;
- можливість додавати опис до кожної роботи;
- оптимізація для мобільних пристроїв;

Корпоративні сайти надають користувачам детальну інформацію про компанію, її продукти та послуги. Вони зазвичай мають складну структуру, велику кількість розділів і розширену навігацію.

Вимоги:

- система керування контентом;
- мультимовність, якщо компанія працює на різних ринках;
- інтеграція з CRM-системами для обробки запитів;
- високий рівень безпеки;
- простий інтерфейс для адміністрування.

Освітні сайти або платформи часто мають модулі для управління курсами, тестами та оцінками. Вони дозволяють студентам реєструватися на курси, проходити тести, отримувати сертифікати. Викладачі можуть створювати завдання, оцінювати студентів.

Вимоги:

- модуль керування курсами;
- підтримка інтерактивних тестів та завдань;
- підтримка мультимедійного контенту;
- логіка користувачів;
- безпека даних користувачів.

Інтернет-магазини мають на меті надання можливості користувачам переглядати товари, додавати їх у кошик, оформляти замовлення та здійснювати оплату онлайн. Вони також повинні мати функціонал для управління товарами, цінами, замовленнями.

Вимоги:

- пошукова система для товарів;
- можливість сортування та фільтрації товарів;
- перегляд карток товарів з описом, цінами, відгуками;
- інтеграція з платіжними системами;
- підтримка кількох способів доставки;
- мобільна версія сайту.

Форум дозволяє користувачам обговорювати різноманітні теми, публікувати повідомлення, створювати нові теми та відповідати на вже існуючі. Він також може включати рейтингову систему та механізми модерації.

Вимоги:

- структурована система категорій і тем;
- підтримка багаторівневих коментарів;
- функція пошуку по форуму;
- модерація контенту;
- інтерфейс для адміністрування;

Соціальні мережі дозволяють користувачам створювати профілі, публікувати пости, спілкуватися через коментарі та особисті повідомлення. Також вони можуть включати можливість додавання фотографій, відео, підписки на інших користувачів.

Вимоги:

- інтерфейс профілю користувача;
- стрічка новин або пости;
- механізм коментарів та вподобань;
- повідомлення та сповіщення;
- інтеграція з іншими сервісами.

Підсумовуючи, можна сказати, що різні типи сайтів мають свої унікальні вимоги та функціональні особливості. Для кожного проекту необхідно правильно підбирати стек технологій, структуру сайту та використовувані інструменти. Вибір типу сайту залежить від цілей, які ставить перед собою його власник. Інтернет-магазини, як один із найбільш складних типів сайтів, вимагатимуть найбільше уваги до функціоналу та технологій, а їх реалізація є дуже важливим етапом у розробці великих проектів.

Інтернет-магазини сьогодні стали невід'ємною частиною онлайн-торгівлі та споживчого досвіду. Вони забезпечують зручність, швидкість та доступність товарів для користувачів по всьому світу. Кожен інтернет-магазин має свої особливості, які дозволяють йому виділятися серед інших, а також застосовує

певні технології для покращення досвіду користувачів [3].

Amazon є одним із найбільших та найвідоміших інтернет-магазинів у світі, що охоплює широкий спектр товарів — від книг до побутової техніки. Однак, що дійсно робить Amazon лідером серед конкурентів, так це його здатність до персоналізації.

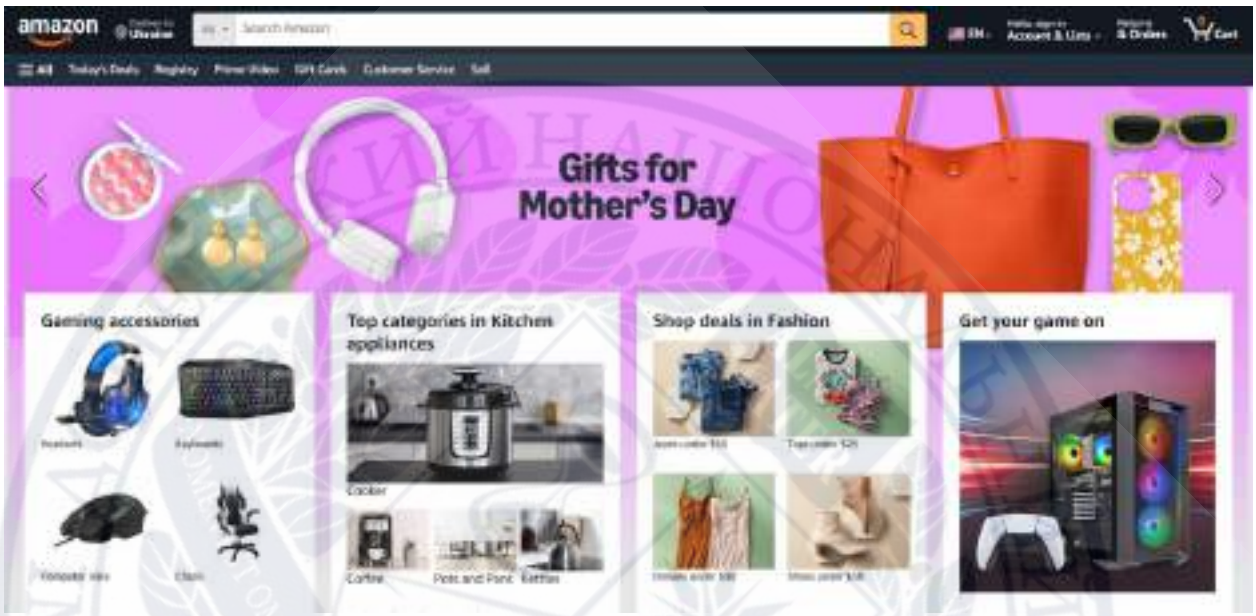

Рисунок 1.1 – Головний екран Amazon

Для персоналізованих рекомендацій Amazon активно використовує алгоритми машинного навчання для надання рекомендацій користувачам, базуючись на їхніх попередніх покупках і переглядах. Інтерфейс сайту є дуже зручним і зрозумілим, що дозволяє швидко знайти необхідний товар, завдяки продуманій системі фільтрації та категоризації. Для інтернаціональності, Amazon дозволяє здійснювати покупки з будь-якої точки світу, підтримує декілька мов та валют, а також надає різні варіанти доставки, включаючи Amazon Prime для більш швидкої доставки.

Amazon використовує широкий спектр технологій, щоб підтримувати свою величезну інфраструктуру:

- для масштабованості та безперебійної роботи сайту використовується AWS (Amazon Web Services). За допомогою цієї технології Amazon може обробляти мільйони запитів одночасно;
- для динамічних елементів інтерфейсу, що дозволяють користувачам

здійснювати покупки та фільтрувати товари без перезавантаження сторінки, застосовуються React і Angular;

- машинне навчання і великі дані використовуються для аналізу поведінки користувачів і надання рекомендацій, що значно покращує досвід покупок.

Rozetka — один із найбільших онлайн-ритейлерів в Україні, що пропонує широкий асортимент товарів. Основною особливістю Rozetka є наявність зручного інтерфейсу та багатої функціональності, яка дозволяє користувачам швидко знайти бажаний товар і оформити покупку. Крім того, мобільний додаток Rozetka дозволяє користувачам здійснювати покупки на ходу, і мобільна версія синхронізується з веб-сайтом, забезпечуючи безперервний досвід.

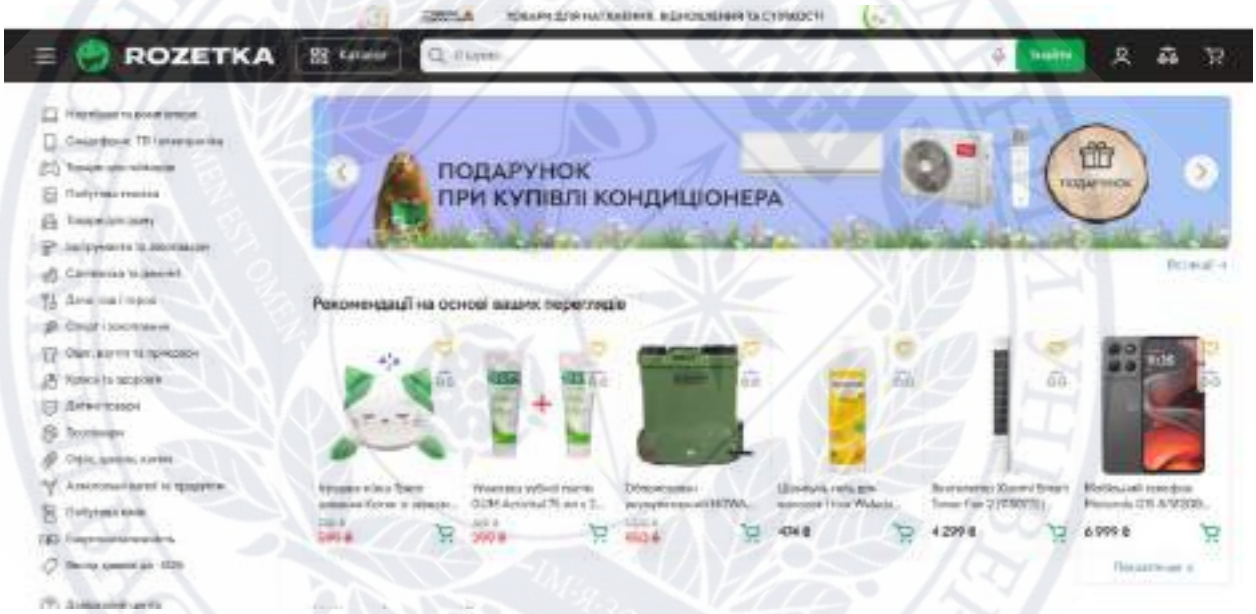

Рисунок 1.2 – Головний екран Rozetka

Rozetka використовує такі технології:

- для серверної логіки та бази даних, що забезпечують швидку обробку запитів, застосовуються PHP і MySQL;
- для динамічних елементів на сайті, що дозволяють реалізувати інтуїтивно зрозумілу навігацію та фільтрацію товарів, використовуються React/Angular;
- Google Analytics використовується для збору даних про поведінку користувачів і оптимізації сайту.

Punisher — це інтернет-магазин, орієнтований на військову тематику, що

спеціалізується на продажу тактичного одягу, спорядження та аксесуарів для фанатів військової культури. З особливостей, простота навігації забезпечує швидкий пошук товарів, незважаючи на специфіку асортименту.

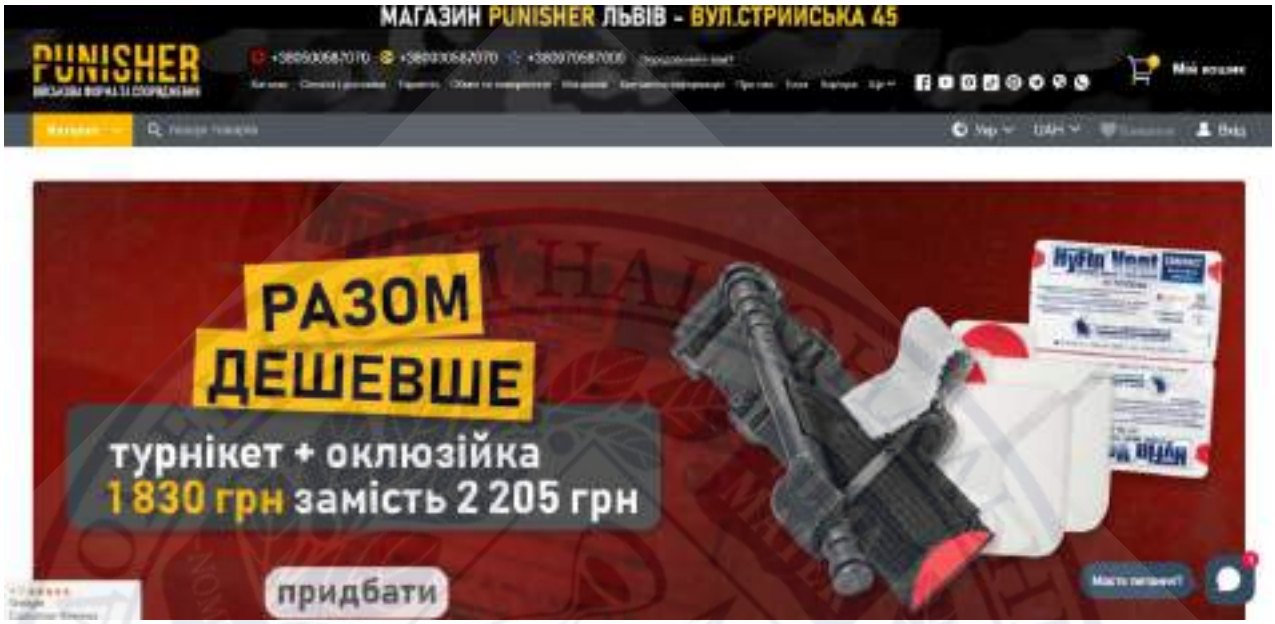

Рисунок 1.3 – Головний екран Punisher

Punisher використовує технології:

- для керування асортиментом та категоріями товарів застосовується Magento, що дає змогу легко оновлювати інформацію про продукти;
- для адаптивного дизайну та інтерактивних елементів сайту використовуються HTML, CSS, JavaScript;
- SEO-оптимізація важлива для залучення трафіку на сайт через пошукові системи.

Prof1group — інтернет-магазин, орієнтований на продаж промислових товарів та обладнання для бізнесу. Магазин спеціалізується на B2B ринку, де продажі здійснюються не лише для індивідуальних споживачів, а й для великих підприємств. На вебсайті, також є система лояльності пропонує знижки та спеціальні умови для постійних клієнтів.

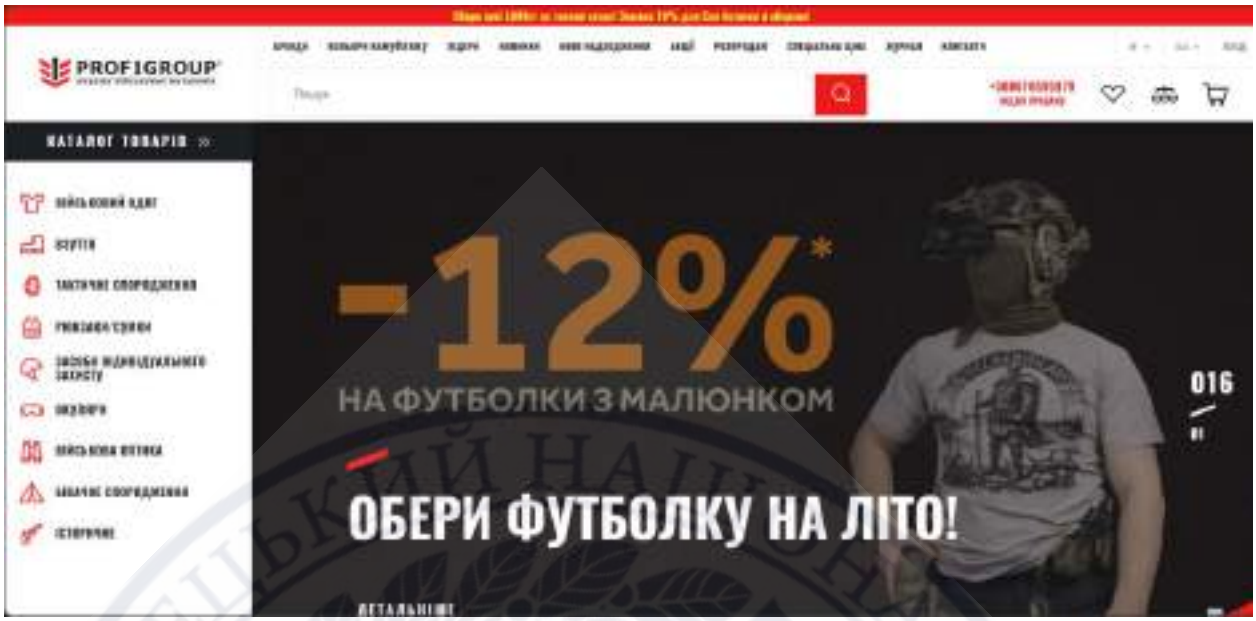

Рисунок 1.4 – Головний екран Prof1group

Prof1group використовує такі технології:

- для управління сайтами та інтеграції з ERP-системами застосовується 1С-Бітрікс;
- для створення адаптивного та динамічного інтерфейсу використовуються HTML5, CSS3, JavaScript;
- Інтеграція з бізнес-процесами для автоматизації обліку замовлень здійснюється через ERP/CRM-системи.

Розглянувши чотири інтернет-магазини — Amazon, Rozetka, Punisher та Prof1group, можна зробити кілька висновків. Власне, кожен з них має свою специфіку, орієнтуючись на різні аудиторії та застосовуючи різні технології для забезпечення зручності користувачів. Важливо, щоб веб-сайт був не тільки технічно потужним, але й інтуїтивно зрозумілим, зручним у користуванні та відповідним потребам своєї цільової аудиторії.

1.2. Основні вимоги до сайту

Перед тим як розпочинати розробку інтернет-магазину військового спорядження, варто чітко зрозуміти, для кого ми його створюємо і що саме цей сайт повинен робити. Це важливий етап, без якого складно побудувати справді зручний та ефективний ресурс.

Основна мета такого інтернет-магазину — надати можливість користувачам швидко, зручно та безпечно купувати тактичне спорядження, одяг, аксесуари та інші товари військового призначення. При цьому магазин повинен враховувати специфіку цільової аудиторії — людей, які цінують якість, функціональність і надійність, та часто знають, чого саме шукають [4].

Щоб забезпечити комфортну роботу з сайтом, важливо реалізувати зручну пошукову систему — наприклад, швидкий пошук за ключовими словами та розширену навігацію по категоріях. Для користувачів також буде корисною можливість переглядати товари на головній сторінці, порівнювати характеристики та фільтрувати асортимент за потрібними параметрами.

Серед ключових функцій сайту — надання детальної інформації про кожен товар: технічні характеристики, якісні фотографії, а в ідеалі ще й відео огляди або демонстрації у використанні. Важливо також дати можливість залишати відгуки, ставити оцінки та читати думки інших покупців — усе це допомагає сформувати довіру до товару.

Не можна обійти стороною питання безпеки. Інтернет-магазин повинен захищати персональні дані своїх користувачів. Наприклад, при зберіганні паролів варто використовувати хешування через такі інструменти, як bcrypt, що дозволяє надійно захищати важливу інформацію в базі даних.

Ще один критичний аспект — дизайн. Він повинен бути не просто привабливим, а й відповідати військовій тематиці, відображати дух бренду та водночас бути інтуїтивно зрозумілим. Простота навігації — запорука того, що користувачі не будуть губитися в категоріях і зможуть швидко знайти потрібне спорядження [5].

З огляду на всі ці вимоги, варто заздалегідь продумати структуру сайту. Розробка такого ресурсу — процес складний, який вимагає як технічної грамотності, так і глибокого розуміння потреб своєї аудиторії.

У технічному плані для реалізації інтернет-магазину доцільно використовувати сучасні вебтехнології:

- для створення адаптивного та динамічного інтерфейсу застосовуються HTML5, CSS3, JavaScript;
- для реалізації серверної логіки та роботи з базою даних доцільно використовувати Node.js або його фреймворки, у поєднанні з MongoDB;
- для збереження та захисту конфіденційної інформації, наприклад паролів, впроваджується bcrypt.

Підсумовуючи, розробка інтернет-магазину військового спорядження являє собою завдання, яке потребує комплексного підходу: аналіз потреб цільової аудиторії, чітке формулювання цілей, увага до дизайну й навігації, впровадження сучасних технологій та забезпечення надійної безпеки. Якщо все зробити правильно, сайт зможе не тільки ефективно продавати, а й стати надійним інструментом комунікації між брендом та його клієнтами.

Аналіз і грамотний вибір основних інформаційних об'єктів, а також визначення способів їхньої взаємодії — це один з ключових етапів у створенні будь-якого вебсайту, зокрема інтернет-магазину з продажу військового спорядження. Без чіткої структури і логіки взаємодії між елементами сайту користувачу буде складно орієнтуватися в каталозі, знаходити потрібні товари або оформлювати замовлення.

До основних категорій інформаційних об'єктів, які відіграють важливу роль у роботі сайту, належать [6]:

1. Каталог товарів — центральний елемент інтернет-магазину. Тут представлено всі позиції військового спорядження, доступні для купівлі. Каталог має включати фільтри за категоріями, наприклад, одяг, амуніція, засоби зв'язку, ціною, виробником чи іншими характеристиками, що спрощує пошук.

2. Сторінка товару — окрема сторінка з розширеною інформацією про конкретний товар. Кожен товар супроводжується описом, якісними фото, технічними характеристиками, відгуками та кнопками для додавання в кошик чи переходу до замовлення.

3. Кошик — об'єкт, що дозволяє користувачеві переглянути обрані товари, змінювати їх кількість, видаляти непотрібне та переходити до оформлення замовлення. Кошик має бути доступним з будь-якої сторінки сайту.

4. Сторінка оформлення замовлення — інтерфейс, через який користувач вказує персональні дані, адресу доставки, обирає спосіб оплати та фіналізує покупку. Важливо забезпечити просту, зрозумілу та безпечну форму заповнення.

5. Профіль користувача — розділ для зареєстрованих користувачів, де зберігається історія замовлень, є можливість редагувати особисті дані, змінювати пароль, переглядати статус доставки тощо.

Для організації злагодженої роботи між усіма цими об'єктами використовується база даних, де зберігається інформація про товари, замовлення та користувачів. У проекті планується використання MongoDB як гнучкого та масштабованого рішення для зберігання даних [7].

Наступним кроком після визначення об'єктів є побудова логіки їх взаємодії. Наприклад, щоб користувач міг зробити замовлення, система повинна підтримувати такі функції, як вибір товару, його додавання в кошик, перехід до оформлення, обробку введених даних, збереження замовлення в базу, повідомлення користувача про результат.

Для реалізації цієї логіки на стороні клієнта доцільно використовувати JavaScript та один із сучасних фреймворків — React, Vue або Angular. Вони дозволяють створювати інтуїтивно зрозумілий інтерфейс із швидкою реакцією на дії користувача [8].

У підсумку, правильний підхід до вибору інформаційних об'єктів та побудови їхньої взаємодії прямо впливає на зручність використання сайту, швидкість виконання дій і загальний досвід користувача. Саме тому цьому етапу потрібно приділити максимум уваги ще до початку програмної реалізації.

1.3. Теоретичні основи користувацького інтерфейсу та взаємодії

Після визначення вимог і основних інформаційних об'єктів сайту наступним важливим кроком є розробка ефективного користувацького

інтерфейсу UI, який забезпечуватиме зручну та логічну взаємодію користувача з вебресурсом.

Користувацький інтерфейс — це сукупність усіх візуальних та функціональних елементів, з якими взаємодіє користувач. Він формує перше враження від сайту, а також значною мірою впливає на зручність і швидкість виконання цільових дій, таких як пошук, вибір, купівля тощо. Для інтернет-магазину військового спорядження це означає створення такого інтерфейсу, який буде водночас функціональним, простим і стилістично відповідним тематиці ресурсу.

Основні принципи розробки UI [9]:

1. Зрозумілість — кожен елемент інтерфейсу повинен мати чітке призначення. Користувач має інтуїтивно розуміти, як користуватись меню, кошиком чи формою замовлення.

2. Консистентність — однакові елементи повинні поводитись однаково на всьому сайті. Це стосується кольорів кнопок, розташування навігації, шрифтів, способів відображення повідомлень.

3. Видимість статусу системи — користувач повинен бачити, що його дії призводять до змін, наприклад, додавання товару в кошик супроводжується повідомленням.

4. Гнучкість і ефективність — для досвідчених користувачів мають бути реалізовані скорочені шляхи, наприклад, швидке оформлення через профіль, а новачкам — докладні інструкції.

5. Мінімізація помилок — інтерфейс має запобігати помилкам, а у випадку їх виникнення — чітко пояснювати, що сталося і як це виправити.

Поняття User Experience тісно пов'язане з UI, але має ширше значення. UX — це загальне враження користувача від взаємодії з сайтом: наскільки легко знайти товар, зручно переглядати інформацію, просто оформити замовлення.

Для досягнення позитивного UX інтерфейс повинен:

- мати логічну структуру сторінок від головної до оформлення замовлення;

- бути оптимізованим під різні пристрої, тобто мати адаптивний дизайн;
- забезпечувати швидке завантаження сторінок;
- підтримувати зворотний зв’язок з користувачем, мати підтвердження дій, повідомлення про помилки, зміни статусів замовлень.

Оскільки сайт призначений для користувачів, які часто точно знають, що шукають, важливо зробити інтерфейс максимально орієнтованим на функціональність:

- мінімалізм — менше декоративних елементів, більше корисної інформації;
- велика увага до зручності каталогу — фільтри, сортування, порівняння товарів, швидкий перегляд;
- оптимізовані форми замовлення — мінімум кроків, чіткі поля, автозаповнення.

Інтерфейс вебсайту реалізується за допомогою таких інструментів:

- HTML5 + CSS3 — для створення структури та стилю сторінок;
- javascript — для реалізації динамічних елементів та інтерактиву;
- фреймворки на кшталт React — для створення сучасного, гнучкого та швидкого інтерфейсу.

Застосування сучасного підходу до UI/UX дозволяє не лише підвищити рівень задоволеності користувачів, а й збільшити конверсії, тобто кількість успішних покупок на сайті. Добре продуманий інтерфейс — це не лише зручність, а й комерційна ефективність.

Під час проектування вебсайту одним із ключових етапів є формування логічної структури сайту — ієрархії сторінок та зв’язків між ними. Грамотно побудована ієрархія забезпечує зручну навігацію для користувача, логічну послідовність дій і спрощує подальше масштабування ресурсу.

Ієрархічна структура вебсайту передбачає поділ сторінок на рівні за принципом "від загального до конкретного". На верхньому рівні зазвичай розташована головна сторінка, яка є центральним вузлом навігації. З неї здійснюється перехід до основних розділів: каталог товарів, про компанію,

контактна інформація, авторизація тощо. Кожен із цих розділів у свою чергу може містити підсторінки або динамічні компоненти [10].

Зокрема, каталог товарів поділяється на категорії або дозволяє відкривати детальну сторінку окремого продукту. Цей перехід реалізується через динамічний маршрут із передачею параметру, що забезпечує генерацію вмісту на основі ідентифікатора товару.

Також важливою частиною структури є сторінка кошика, звідки користувач переходить до оформлення замовлення, а після покупки — до перегляду власних замовлень. Це створює послідовну логіку користувацького шляху: вибір товару → перегляд → додавання в кошик → оформлення → підтвердження → перегляд історії.

Особливу роль у навігації виконують навігаційне верхнє меню і футер, які містять посилання на основні сторінки. Крім того, у проекті реалізовано пошуковий рядок, що забезпечує прямий доступ до результатів на основі запиту.

Таким чином, побудова ієрархії сторінок і логіки переходів є не лише структурною основою вебсайту, а й важливою умовою для досягнення високої якості користувацького досвіду.

Сучасний вебсайт не може обмежуватися лише десктопною версією. Більшість користувачів заходять на інтернет-магазини зі смартфонів або планшетів, тому адаптивний дизайн є обов’язковою вимогою до будь-якого сучасного вебресурсу, включаючи магазин військового спорядження.

Адаптивний дизайн — це метод створення інтерфейсу, який автоматично підлаштовується під розмір і тип екрана користувача. Він забезпечує однаково зручне використання сайту як на великому моніторі, так і на маленькому смартфоні.

Основні принципи адаптивного дизайну [11]:

- гнучка сітка flexbox, grid замість фіксованих піксельних розмірів;
- медіазапити для зміни стилів залежно від ширини екрана;
- відносні одиниці вимірювання замість абсолютних;
- мобільне меню та спрощене відображення деяких блоків.

Для магазину тактичного спорядження це означає:

- зручний пошук товарів на смартфоні;
- швидкий перегляд фото та характеристик навіть у польових умовах;
- просте оформлення замовлення з мобільного без потреби масштабувати екран.

Для реалізації адаптивності й доступності в проекті використовуються такі технології:

- CSS Flexbox та Grid Layout — гнучке компонування блоків;
- media queries — перебудова макета залежно від ширини вікна;
- фреймворки на зразок Tailwind CSS або Bootstrap — прискорюють розробку адаптивного інтерфейсу;
- lighthouse, Axe, Wave — інструменти для автоматичної перевірки доступності сайту.

У підсумку, адаптивний дизайн і забезпечення доступності — це не додаткові опції, а критично важливі складові якісного вебресурсу. Їх ігнорування призводить до втрати частини потенційних клієнтів, тоді як їх грамотна реалізація дозволяє сайту працювати ефективно на всіх пристроях і для всіх користувачів.

У процесі проектування користувацького інтерфейсу інтернет-магазину військового спорядження критично важливо не лише реалізувати необхідні функціональні можливості, але й уникнути розповсюджених помилок, які можуть значно знизити ефективність ресурсу та погіршити загальне враження користувача. Зниження рівня зручності або порушення логіки взаємодії із сайтом, навіть на незначному етапі, здатне призвести до втрати потенційних клієнтів [12].

Однією з найпоширеніших помилок є нечітка або перевантажена навігаційна система. Якщо структура категорій товарів є нелогічною, а користувачеві складно зрозуміти, де шукати потрібну продукцію, зростає ймовірність його відмови від подальшої взаємодії з сайтом. Вирішенням цієї проблеми є впровадження чітко структурованої системи навігації з можливістю

фільтрації товарів за категоріями, ціною, призначенням тощо. Додатково варто забезпечити наявність зручного пошуку за ключовими словами, а також передбачити “хлібні крихти”, які допомагають орієнтуватися в ієрархії сторінок.

Іншою проблемою є перевантаження інтерфейсу зайвими елементами — банерами, кнопками, рекламними блоками або надмірною кількістю тексту. Подібне перевантаження негативно впливає на сприйняття інформації та відволікає увагу користувача від основних дій. Для уникнення цієї помилки доцільно застосовувати принципи мінімалізму: залишати лише найнеобхідніші елементи, використовувати контрастні шрифти й відступи для логічного групування інформації, а також забезпечити візуальну ієрархію за допомогою кольору, розміру та розташування компонентів.

Негативно на досвід користувача також впливає відсутність повної та достовірної інформації про товари. Короткий опис, неякісні зображення, відсутність технічних характеристик або відгуків зменшують довіру покупця та заважають прийняттю рішення. Для вирішення цього питання кожен товар має бути представлений детальним описом, кількома фотографіями з різних ракурсів, специфікаціями, а також відгуками інших покупців, які формують соціальне підтвердження якості продукції.

Окремої уваги потребує процес оформлення замовлення. Якщо він занадто складний або вимагає великої кількості дій, наприклад, обов’язкова реєстрація перед купівлею, це може стати бар’єром для завершення покупки. Щоб цього уникнути, необхідно впровадити можливість “гостьового” замовлення без створення акаунту, забезпечити зрозуміле групування полів введення даних, а також автоматичні підказки й перевірку коректності введення.

Поширеною помилкою також є відсутність зворотного зв’язку на дії користувача. Наприклад, після натискання кнопки “додати до кошика” інтерфейс не надає жодного візуального підтвердження. У таких випадках користувач може сумніватися, чи відбулася дія, або спробувати повторити її, що призводить до плутанини. Вирішенням є впровадження повідомлень, анімацій, зміни кольору або інших візуальних ефектів, які підтверджують виконання дії.

Ще одним критичним аспектом є адаптація сайту до мобільних пристроїв. В умовах сучасного використання переважно смартфонів, відсутність адаптивної верстки або складне керування в мобільній версії призводить до втрати значної частки користувачів. Тому необхідно забезпечити адаптивний дизайн з відповідним масштабуванням елементів, простотою навігації та тестуванням відображення інтерфейсу на різних типах пристроїв.

Загалом, ігнорування очікувань користувачів та стандартів поведінки в інтерфейсах, наприклад, розташування кнопки “кошик” у незвичному місці або нестандартна поведінка кнопки “назад” створює додаткові труднощі у взаємодії з сайтом. Тому важливо дотримуватися вже усталених шаблонів користувацького інтерфейсу, які є інтуїтивно зрозумілими для більшості аудиторії.

Таким чином, системний аналіз можливих помилок у користувацькому досвіді та своєчасне їх уникнення є обов’язковою складовою якісної реалізації інтерфейсу інтернет-магазину. Успішне вирішення зазначених проблем забезпечує не лише комфорт користувача, але й підвищення рівня довіри до ресурсу, що безпосередньо впливає на кількість успішних замовлень.

1.4. Поняття безпеки вебресурсів

У сучасному інформаційному просторі, де інтернет став невіддільною складовою щоденної діяльності як приватних осіб, так і організацій, питання веббезпеки набуває особливої актуальності. Зростання обсягів переданої та оброблюваної через вебсайти інформації, зокрема персональних даних, платіжних реквізитів, комерційної інформації, потребує належного захисту від несанкціонованого доступу, втрати, модифікації або зловмисного використання.

Поняття веббезпеки охоплює систему заходів, спрямованих на забезпечення захищеності вебресурсів від загроз, які можуть виникати в процесі їх функціонування. До таких заходів належать як програмно-технічні рішення, так і організаційні політики та процедури, що мають на меті гарантування трьох

ключових властивостей інформації: конфіденційності, цілісності та доступності [13].

Роль веббезпеки у функціонуванні інтернет-ресурсу є багатогранною:

- захист персональних даних користувачів. Згідно з вимогами міжнародного та національного законодавства, зокрема Загального регламенту про захист даних GDPR та Закону України «Про захист персональних даних», вебресурс має гарантувати безпечне зберігання та обробку чутливої інформації;
- забезпечення довіри користувачів. Наявність механізмів автентифікації, використання шифрування трафіку, наприклад, протокол HTTPS, систем сповіщення про активність облікових записів тощо сприяє підвищенню рівня довіри до ресурсу;
- збереження репутації вебресурсу. Будь-який інцидент, пов’язаний з порушенням безпеки, наприклад, злам акаунтів або витік даних, негативно впливає на імідж компанії та може призвести до втрати клієнтів і фінансових збитків;
- надійність функціонування системи. Веббезпека дозволяє запобігти технічним збоям, що можуть виникнути внаслідок дій зловмисників — таких як DDoS-атаки, SQL-ін’єкції, міжсайтові скрипти тощо;
- виконання нормативних вимог. Упровадження політик безпеки є необхідною умовою для відповідності законодавству, що регламентує діяльність у сфері електронної комерції та захисту інформації.

У контексті електронної комерції, зокрема при розробці інтернет-магазину військового спорядження, питання веббезпеки набуває ще більшої ваги. Окрім захисту особистих і платіжних даних користувачів, такий ресурс повинен враховувати специфіку продукції, що може накладати додаткові вимоги до систем автентифікації, логування дій користувачів, контролю доступу до певного функціоналу [14].

Таким чином, веббезпека виступає не лише засобом технічного захисту, але й інструментом формування довготривалих відносин між користувачем і

платформою. Вона забезпечує стабільність і безперервність функціонування системи, підвищує лояльність аудиторії та сприяє сталому розвитку вебресурсу.

У сучасному цифровому середовищі, де вебресурси стають об'єктом постійних атак, важливу роль відіграє дотримання міжнародних стандартів та нормативів у сфері інформаційної безпеки. Вони встановлюють чіткі вимоги та рекомендації щодо безпечного проектування, розробки, експлуатації та моніторингу вебдодатків і пов'язаних з ними інформаційних систем.

Одним із ключових документів у цій галузі є стандарт ISO/IEC 27001, що визначає вимоги до створення, впровадження, підтримки та постійного вдосконалення системи управління інформаційною безпекою. Він забезпечує уніфікований підхід до захисту конфіденційної інформації, управління ризиками та документування процесів безпеки. Його впровадження дозволяє організаціям системно підходити до безпеки вебресурсів та отримати міжнародне визнання якості відповідних заходів.

Також важливим джерелом практичних рекомендацій є Open Web Application Security Project — некомерційна організація, яка публікує рейтинг найбільш критичних вразливостей вебдодатків у вигляді списку OWASP Top 10. Цей документ є своєрідним галузевим стандартом, який визначає найбільш поширені проблеми безпеки, зокрема, ін'єкції, вразливості автентифікації, неправильну конфігурацію безпеки та містить рекомендації щодо їх усунення. Дотримання принципів OWASP є обов'язковою практикою у безпечній розробці сучасних вебрішень.

Іншим вагомим нормативним документом є RFC 7525, що регламентує використання криптографічних протоколів у сучасних системах, зокрема Transport Layer Security, для забезпечення захищеної передачі даних у вебмережах. Цей стандарт вказує на необхідність відмови від застарілих та уразливих версій протоколів, наприклад, SSLv3 або TLS 1.0 на користь більш безпечних конфігурацій.

Крім міжнародних стандартів, у деяких випадках вебресурси повинні відповідати й галузевим або національним вимогам, зокрема:

- GDPR — загальноєвропейський регламент, що регулює захист персональних даних користувачів. Вебресурси, що працюють з особистою інформацією громадян ЄС, повинні дотримуватись вимог щодо конфіденційності, прозорості обробки даних та згоди користувачів;
- закон України “Про захист персональних даних” — регламентує порядок збирання, зберігання, обробки та використання персональної інформації вітчизняними вебресурсами. Його дотримання є обов’язковим для уникнення адміністративної та кримінальної відповідальності;
- Payment Card Industry Data Security Standard — стандарт для компаній, які працюють із платіжними картками. Він встановлює вимоги до захисту платіжних даних у вебсередовищі.

Впровадження зазначених стандартів і дотримання нормативів є не лише засобом забезпечення належного рівня інформаційної безпеки, а й чинником, що формує довіру користувачів до вебресурсу, підвищує його репутацію та сприяє конкурентоспроможності в умовах сучасного ринку.

У контексті створення вебресурсу для електронної комерції, зокрема інтернет-магазину, одним із ключових аспектів, що впливає як на стабільність роботи системи, так і на довіру користувачів, є реалізація ефективної моделі безпеки. Інтернет-магазини оперують великим обсягом конфіденційної інформації, включаючи персональні дані клієнтів, облікові записи, історію покупок та платіжну інформацію. Тому в межах розробки такого ресурсу доцільно впроваджувати комплекс заходів, спрямованих на попередження потенційних загроз.

Загрози, що найчастіше зустрічаються в межах електронної комерції, охоплюють: перехоплення сеансів користувачів, викрадення облікових даних, атаки типу SQL-ін’єкція, міжсайтове скриптування, підміна запитів, а також атаки, спрямовані на компрометацію серверного середовища або бази даних [15].

Для нейтралізації зазначених ризиків у практиці безпечної розробки використовуються такі підходи [16]:

1. Одним із базових принципів є відмова від зберігання паролів у відкритому вигляді. У типовому рішенні застосовується алгоритм хешування паролів із додаванням "солі" — зокрема, широко поширеним є використання бібліотеки bcrypt, яка дозволяє забезпечити стійкість до атак типу перебору та використання попередньо сформованих хешів.

2. Використання маркерів доступу. Для організації системи автентифікації на стороні клієнта часто використовується підхід із застосуванням JSON Web Token, що дозволяє реалізувати безпечне збереження інформації про сесію користувача без потреби у постійному зверненні до бази даних. Токен кодує необхідну інформацію й передається з кожним запитом, проходячи верифікацію на стороні сервера.

3. Ізоляція конфіденційної інформації через змінні середовища. Ключі доступу до баз даних, секрети шифрування, а також конфігураційні параметри сервісів повинні бути винесені у спеціальні файли середовищ та недоступні для зовнішнього оточення. Це дозволяє знизити ризик випадкового витоку інформації через публікацію вихідного коду або помилки у конфігурації сервера.

4. Забезпечення шифрування трафіку через HTTPS. Наявність дійсного SSL/TLS-сертифіката на вебсайті є обов’язковою умовою для захисту переданих даних між клієнтом і сервером. Протокол HTTPS гарантує автентичність ресурсу, шифрування сесії та цілісність переданих повідомлень.

5. Контроль доступу та рольова модель. Впровадження ролей користувачів забезпечує розмежування прав доступу до функціоналу вебсайту. Наприклад, адміністратор має право змінювати вміст сайту та переглядати статистику, тоді як пересічний покупець — лише здійснювати покупки та переглядати замовлення.

6. Логування та аудит активності. Журнали подій є важливим інструментом для виявлення підозрілої активності. Логування спроб входу, помилок авторизації, змін у критичних розділах сайту дає змогу оперативно реагувати на інциденти та здійснювати постінцидентний аналіз.

7. Оновлення бібліотек та залежностей. Регулярна перевірка й оновлення сторонніх залежностей дає змогу уникнути використання вразливих версій бібліотек, які можуть містити відомі експлойти.

Застосування вказаних заходів є частиною стандартної стратегії Security by Design, коли безпека розглядається не як додатковий компонент, а як базовий принцип розробки вебсайту. У результаті реалізований інтернет-магазин функціонує не лише як інструмент торгівлі, а й як захищене середовище для взаємодії з користувачами, що відповідає сучасним вимогам до електронної комерції.

РОЗДІЛ 2
ПОСТАНОВКА ЗАДАЧІ ТА ВИБІР ІНСТРУМЕНТІВ

2.1. Постановка задачі

Основною задачею вебдодатку є реалізація повноцінного інтернет-магазину з усіма необхідними функціями для роботи з товарами, користувачами, замовленнями, кошиком та обробкою оплати. Для цього має використовуватись відповідна база даних, яка має зберігати, обробляти та передавати інформацію у відповідні частини інтерфейсу [17].

Перш за все, кожен товар має мати такі характеристики: назву, опис, ціну, фотографію або декілька фотографій, категорію, підкатегорію, доступні розміри, а також дату додавання товару. Уся ця інформація має зберігатися в базі даних і передаватися на сайт для відображення у картках товарів, на детальній сторінці продукту та в кошику. Назва, опис та фото мають відображатися в точності так, як вони записані в базі. Ціна має використовуватись для обрахунку підсумкової вартості замовлення.

Друга ключова задача – це реалізація системи реєстрації та входу користувача. У базі даних має зберігатись інформація про кожного користувача: його ім’я, електронна пошта, пароль, а також спеціальне поле, яке містить дані кошика користувача. Користувач, що не пройшов авторизацію, має мати змогу лише переглядати товари та переходити по сторінках, але не має мати доступу до основних функцій вебсайту, таких як додавання товарів у кошик, оформлення замовлень чи перегляд історії покупок. У хедері сайту в такому випадку мають відображатися лише кнопки входу та реєстрації.

Після реєстрації користувач має мати можливість додавати обрані товари до кошика, переглядати та змінювати його вміст, а також має мати доступ до історії власних замовлень. В історії замовлень користувач має мати змогу переглядати інформацію про придбані товари та оновлювати сторінку для перевірки актуальної інформації, яку міг змінити адміністратор.

Кожен обліковий запис має мати додаткове поле — статус адміністратора. Якщо користувач є адміністратором, він має мати доступ до окремої панелі керування в інтерфейсі сайту. У цій панелі він має мати можливість додавати нові товари із зазначенням назви, опису, ціни, фотографій, розмірів, категорій і підкатегорій. Також адміністратор має мати можливість змінювати або видаляти товари, переглядати замовлення користувачів і редагувати інформацію про них.

Ще однією ключовою задачею є створення системи замовлень. Кожне замовлення має містити ID користувача, список товарів, загальну суму покупки, адресу доставки, спосіб оплати, статус замовлення (наприклад, «Очікує підтвердження» або «Доставлено»), інформацію про здійснення оплати, а також дату створення замовлення у форматі Unix. Уся ця інформація має використовуватись для відображення історії покупок користувача, а також для обробки замовлень у панелі адміністратора.

Не менш важливою є функціональність кошика. Авторизований користувач має мати змогу додавати товари до кошика, змінювати їх кількість, а також видаляти непотрібні позиції. Кошик має мати динамічну частину з рахунком, де користувач може переглядати загальну вартість вибраних товарів, бачити розрахунок кожної одиниці товару за ціною та кількістю, а також отримувати автоматичне оновлення загальної суми при зміні вмісту кошика.

Також вебсайт має мати реалізовану пошукову систему. Кожен користувач, навіть неавторизований, має мати можливість скористатись пошуком для знаходження товарів. Пошукова система має враховувати як точну назву товару, так і часткові збіги. Додатково має бути реалізована можливість пошуку за категоріями і підкатегоріями, які призначаються адміністратором, а також сортування товарів за ціною — від найдешевших до найдорожчих і навпаки.

Для забезпечення належного рівня безпеки користувачів, а також захисту доступу до персональних і адміністративних функцій сайту, має бути реалізовано сучасні механізми авторизації та шифрування.

Зокрема, система має використовувати бібліотеку bcryptjs для хешування паролів користувачів. Це означає, що при створенні акаунта або зміні паролю введене значення має автоматично хешуватись і зберігатися у базі даних у вигляді зашифрованого хешу, який неможливо зворотно розшифрувати. Під час входу користувача система має перевіряти правильність введеного пароля шляхом порівняння збереженого хешу з новим, створеним на основі введеного значення.

Окрім цього, після успішної авторизації користувачу має видаватися токен доступу у вигляді JWT (JSON Web Token), який містить зашифровану інформацію про користувача: його ID, статус (звичайний користувач або адміністратор) та термін дії. Цей токен має зберігатися у браузері (наприклад, у локальному сховищі або HTTP-куках) та автоматично додаватись до кожного запиту до захищених маршрутів. Система має мати механізм перевірки дійсності токена на сервері, і, якщо токен недійсний або прострочений, користувач має бути автоматично перенаправлений на сторінку входу.

У разі, якщо токен містить інформацію про адміністратора, користувач має мати змогу отримати доступ до панелі керування та користуватись відповідними адміністративними функціями. Усі обмеження мають контролюватися на основі інформації, яка міститься в JWT-токені.

Підсумовуючи, поставлені задачі охоплюють усі ключові аспекти роботи сучасного інтернет-магазину: від структури товару й реєстрації користувача до обробки замовлень, забезпечення безпеки та адміністрування. Це має дозволити побудувати повноцінну, масштабовану платформу для онлайн-покупок і онлайн-продажів із належним рівнем доступу, функціональності та захисту даних користувачів.

2.2. Огляд інструментів створення вебдодатку

Вебдодаток — це тип програмного забезпечення, який функціонує у браузері без потреби встановлення на комп’ютер користувача. Основним середовищем для доступу до веб-додатка є веб-браузер, наприклад, Chrome,

Mozilla Firefox, Opera тощо. Користувач, або клієнт, взаємодіє з додатком через графічний інтерфейс, який розробляється за допомогою мов HTML, CSS і JavaScript, що підтримуються сучасними браузерами [18].

Основний принцип роботи вебдодатку полягає у взаємодії клієнтської та серверної сторін через мережу Інтернет. Клієнтська частина відповідає за відображення інформації та взаємодію з користувачем, тоді як серверна сторона опрацьовує запити, виконує логіку додатку і керує базою даних. Обмін даними між клієнтом і сервером відбувається за допомогою HTTP-протоколу.

Коли користувач звертається до веб-додатку через браузер, відправляється HTTP-запит до сервера. Цей запит може містити дані, введені користувачем, або вимогу отримати певну сторінку. Серверна сторона, яка складається з бекенд-фреймворків, веб-сервера та бази даних, обробляє запит, виконує необхідні обчислення або пошук даних, після чого формує відповідь. У відповідь надсилається HTML-код разом із CSS-стилями, JavaScript-скриптами та іншими ресурсами, наприклад зображення, шрифти тощо, які браузер відображає у вигляді веб-сторінки [19].

Крім цього, веб-додатки дозволяють реалізовувати інтерактивні дії. Наприклад, користувач має змогу вводити дані в форми, натискати кнопки, змінювати налаштування тощо. Кожна така взаємодія може ініціювати новий запит до сервера, після якого відбувається оновлення вмісту сторінки.

У контексті інтернет-магазину, веб-додаток має реалізовувати наступні принципи [20]:

- користувач має мати можливість переглядати товари, додавати їх до кошика, оформляти замовлення та здійснювати оплату;
- сервер має обробляти запити користувача, надаючи актуальну інформацію про товари, їх наявність та ціни;
- клієнтська частина має відображати відповідні інтерфейсні елементи відповідно до отриманих даних — картки товарів, підсумкову вартість замовлення, статуси тощо;

- система має забезпечувати збереження даних про сесії користувача, історію замовлень та облікові записи;
- Комунікація між клієнтом і сервером має відбуватись швидко, безпечно і безперервно, що особливо важливо при авторизації та оплаті.

Таким чином, принцип роботи веб-додатку базується на постійному циклі "запит — обробка — відповідь", який забезпечує динамічну взаємодію користувача з додатком через браузер, надаючи зручність, масштабованість та доступність з будь-якого пристрою.

Веб-додаток інтернет-магазину складається з низки ключових компонентів, кожен з яких виконує певну функцію та забезпечує безперервну роботу сервісу. Ці компоненти тісно взаємодіють між собою, утворюючи єдину систему, яка дозволяє реалізовувати повноцінну логіку електронної комерції. Основні компоненти веб-додатку інтернет магазину [21]:

1. Фронтенд — це візуальна частина веб-додатку, з якою безпосередньо взаємодіє користувач через браузер. Він забезпечує відображення сторінок, а також зручну та зрозумілу навігацію сайтом. Основними технологіями фронтенду є HTML, CSS і JavaScript. Для розширення функціональності можуть використовуватись сучасні фреймворки React, Vue, Angular, які дозволяють створювати динамічні інтерфейси. Через фронтенд реалізується пошук товарів, додавання в кошик, оформлення замовлення тощо.

2. Бекенд відповідає за обробку логіки додатку, взаємодію з базою даних, авторизацію користувачів, перевірку введених даних та інші серверні операції. Саме бекенд приймає запити, що надходять з клієнтської частини, опрацьовує їх і повертає відповідь. Для створення бекенду можуть використовуватись різні технології, наприклад: Node.js, PHP, Python, а саме його бібліотеки Django і Flask, Java, тощо.

3. База даних виконує роль сховища інформації, необхідної для роботи інтернет-магазину. Вона містить таблиці з даними про товари, користувачів, замовлення, оплату та доставку. Бази даних можуть бути реляційними, такі як MySQL та PostgreSQL або нереляційними, така як MongoDB, в залежності від

специфіки реалізації проекту. Взаємодія між бекендом і базою даних дозволяє зчитувати, змінювати або зберігати інформацію у відповідь на запити користувача.

4. Application Programming Interface (API) є посередником між клієнтською частиною і серверною частиною. Через API фронтенд надсилає запити, наприклад, отримати список товарів, оформити замовлення, а бекенд обробляє ці запити й повертає відповіді у форматі JSON або XML. API можуть бути реалізовані у вигляді REST, що є найпоширенішим варіантом, GraphQL або gRPC. Для створення REST API використовують фреймворки Express.js для Node.js, Django REST Framework для Python, Spring Boot для Java тощо. Саме API дозволяє безпечно й ефективно читати дані з бази та записувати їх туди, із чітким поділом відповідальності між клієнтом і сервером.

5. Важливою частиною будь-якого інтернет-магазину є інтеграція із платіжними сервісами, наприклад, Stripe, LiqPay, PayPal, які дозволяють проводити транзакції в режимі онлайн. Крім того, бекенд взаємодіє із сервісами доставки, передаючи дані про замовлення, адреси користувачів та статуси виконання доставки. Надійна інтеграція цих компонентів забезпечує швидке та безпечне завершення покупки.

Усі вищезгадані компоненти працюють разом, формуючи повний цикл взаємодії користувача з інтернет-магазином. Наприклад, коли покупець додає товар до кошика, ця інформація обробляється бекендом та зберігається у базі даних, а на екрані миттєво оновлюється інтерфейс за допомогою фронтенду. У процесі оформлення замовлення підключаються компоненти оплати та доставки, а система аналітики фіксує відповідні події для подальшого аналізу.

Таким чином, комплексне функціонування фронтенду, бекенду, бази даних, платіжних та аналітичних сервісів забезпечує повноцінну роботу веб-додатку інтернет-магазину. Це дозволяє створити зручний, безпечний та ефективний торговий майданчик як для користувачів, так і для адміністраторів ресурсу.

Розробка веб-сайтів — це складний процес, що включає різноманітні етапи, від планування архітектури до реалізації функціональності і забезпечення взаємодії з користувачем. Щоб успішно реалізувати цей процес, розробники використовують цілу низку інструментів, кожен з яких виконує специфічні функції та забезпечує максимальну ефективність роботи. Важливою є також інтеграція цих інструментів між собою, щоб створити гармонійну систему, що працює на задоволення вимог користувачів та замовників [22].

Основою для розробки веб-сайтів є мови програмування, які розрізняються за своїми функціями та сферами застосування. Для створення інтерфейсів користувача, які є основною частиною фронтенду, застосовуються:

- javaScript — мова, яка дозволяє створювати динамічні елементи сторінки, обробляти події, реалізовувати анімації та забезпечувати взаємодію з сервером без перезавантаження сторінки. У поєднанні з HTML і CSS вона утворює основу будь-якого інтерфейсу. Окрім того, завдяки можливості запускати JavaScript на сервері через Node.js, ця мова стала універсальною для створення як клієнтської, так і серверної частини веб-додатків [23];

Для серверної частини використовуються також інші мови, зокрема:

- python з фреймворком Django — забезпечує високу продуктивність і простоту розробки, зручний для створення веб-додатків і REST API;
- PHP — традиційно застосовується для серверної логіки, особливо в контексті популярних CMS, таких як WordPress;
- ruby з фреймворком Ruby on Rails — ідеальний для швидкої розробки веб-додатків завдяки своїй гнучкості та великій кількості готових рішень;
- java та C# — використовуються для масштабованих корпоративних рішень, де важлива стабільність і висока пропускна здатність.

Фреймворки та бібліотеки допомагають розробникам уникнути необхідності «винаходити велосипед», дозволяючи використовувати готові рішення для типових завдань. До найпопулярніших фреймворків та бібліотек належать [24]:

- express.js — фреймворк для Node.js, що дає змогу створювати веб-сервери і API за допомогою мінімуму коду. Він є основою для створення REST API.
- django — високорівневий фреймворк для Python, який допомагає швидко реалізувати складні веб-додатки з високим рівнем безпеки та підтримкою багатьох вбудованих функцій;
- Ruby on Rails — фреймворк, що автоматизує багато процесів розробки, дозволяючи значно скоротити час створення прототипу;
- laravel — фреймворк для PHP, що забезпечує потужні функції для роботи з базами даних і маршрутизацією, а також зручний механізм аутентифікації.

На клієнтській стороні застосовуються такі потужні інструменти:

- react — бібліотека JavaScript для побудови складних інтерфейсів користувача. Вона дозволяє створювати компоненти, які оновлюються лише при зміні даних, що забезпечує високу швидкість і зручність у використанні;
- vue.js — легкий і гнучкий фреймворк, який дозволяє створювати інтерфейси з мінімумом налаштувань і високою продуктивністю;
- angular — потужний фреймворк від Google, який підходить для великих проектів і забезпечує всі необхідні інструменти для побудови складних веб-додатків.

Для ефективного зберігання і обробки даних веб-додатки зазвичай використовують реляційні та нереляційні бази даних:

- MySQL — одна з найпоширеніших реляційних СУБД, що підтримує складні запити, транзакції та високу продуктивність;
- PostgreSQL — потужна реляційна СУБД з підтримкою складних запитів і транзакцій, що використовується в великих проектах, де важлива стабільність і точність;
- MongoDB — документоорієнтована база даних, яка забезпечує високу гнучкість у зберіганні даних, що особливо корисно при роботі з непостійною схемою даних.

Вибір бази даних залежить від специфіки проєкту: для більш структурованих даних підходять реляційні СУБД, тоді як для великої кількості варіативних і складних даних — нереляційні.

Керування версіями коду — важливий аспект командної роботи при розробці веб-додатків. Git є стандартом індустрії для відслідковування змін у коді. Завдяки йому можна працювати над різними гілками проєкту, що дозволяє здійснювати паралельну розробку без ризику перезапису даних [25].

Також популярними є платформи для зберігання кодів в репозиторіях: GitHub, GitLab і Bitbucket. Ці сервіси дозволяють легко організувати командну роботу, підтримують інтеграцію з CI/CD системами для автоматичного тестування і розгортання коду [26].

Веб-сервери, такі як Nginx та Apache, відповідають за обробку запитів від клієнтів і перенаправлення їх до відповідних сервісів або ресурсів. Вони є основою для роботи будь-якого веб-додатка в інтернеті.

Кожен з перелічених інструментів відіграє важливу роль у створенні веб-додатків, і їх вибір залежить від специфіки проєкту та вимог замовника. Однак правильно поєднуючи ці інструменти та технології, можна створити потужний і надійний веб-додаток, що відповідає всім сучасним вимогам до функціональності та продуктивності.

2.3. Моделі вебсайту

Інформаційна архітектура вебсайту є однією з ключових складових у процесі розробки сучасних цифрових продуктів. Вона охоплює принципи організації, структурування та подання інформації таким чином, щоб користувачі могли легко орієнтуватися на сайті, знаходити необхідні відомості та ефективно взаємодіяти з контентом [27].

У найширшому розумінні, інформаційна архітектура — це дисципліна, яка забезпечує логічну побудову навігації та структури вебресурсу відповідно до потреб користувача та цілей бізнесу. Вона включає розробку ієрархії сторінок,

системи навігації, категоризації, а також визначення логіки переходу між елементами інтерфейсу.

До основних компонентів інформаційної архітектури належать:

- ієрархічна структура сайту, яка визначає взаємозв'язки між основними та другорядними сторінками;
- система навігації, що забезпечує швидкий доступ до потрібних розділів, наприклад меню, кнопки, вкладки, посилання;
- пошукові та фільтраційні інструменти, які допомагають користувачеві ефективно знаходити інформацію або товари;
- контентна класифікація, що полягає у групуванні матеріалів за категоріями, тегами або іншими логічними ознаками;
- візуальна організація інформації, яка передбачає використання зрозумілої і послідовної візуальної мови: шрифтів, кольорів, блоків, кнопок тощо.

Правильно спроектована інформаційна архітектура відіграє критично важливу роль у досвіді користувача. Вона не лише спрощує навігацію, а й створює відчуття впорядкованості та логічності, що вкрай важливо для зручності, особливо в інтернет-магазинах, де користувач щоденно взаємодіє з великою кількістю інформації: товарами, фільтрами, кошиком, замовленнями тощо.

З погляду розробника, інформаційна архітектура виступає фундаментом, на основі якого будується вся технічна реалізація вебресурсу. Саме вона визначає, яку модель проектування варто обрати, які технології будуть найбільш доцільними, а також як взаємодіятимуть компоненти сайту між собою. У наступному підрозділі буде розглянуто моделі архітектури вебзастосунків, серед яких обрано ту, що найкраще відповідає цілям та завданням проєкту [27].

У процесі створення вебсайтів важливо обрати відповідну модель архітектури застосунку. Від цього залежить масштабованість, підтримуваність, зручність розробки та ефективність взаємодії між окремими компонентами сайту. Існує кілька поширених моделей, які обираються залежно від типу

застосунку, обсягу функціональності та обраних технологій. До таких моделей можу виокремити:

1. Модель Model–View–Controller (MVC) – одна з найпоширеніших моделей архітектури, яка чітко поділяє логіку додатку на три компоненти:

- model — відповідає за управління даними, включаючи взаємодію з базою даних;
- view — відображає інформацію користувачеві. Це HTML, CSS, динамічні шаблони тощо;
- controller — обробляє запити, взаємодіє з моделлю, і передає дані до представлення.

MVC ідеально підходить для застосунків, у яких багато запитів до бази даних і чітко розділена серверна й клієнтська логіка. Часто використовується у фреймворках Laravel, Django, Ruby on Rails.

2. Модель Model–View–ViewModel (MVVM), ця модель, подібна до MVC, проте більш орієнтована на двосторонню прив'язку даних, що робить її зручною для інтерактивних інтерфейсів. ViewModel виступає як посередник між View і Model, обробляючи зміну стану в реальному часі. MVVM популярна у Angular, Vue.js, Knockout.js.

3. Компонентна архітектура – це сучасна архітектурна модель, яка є основою для розробки Single Page Applications (SPA) додатків за допомогою React, Vue, Svelte тощо. У цій моделі логіка поділяється на незалежні компоненти, кожен із яких відповідає за конкретну частину інтерфейсу, наприклад, навігація, сторінка товару, кошик тощо.

Переваги:

- висока гнучкість та модульність;
- повторне використання коду;
- зручне управління станом;
- добра підтримка великих проектів із багатьма сторінками.

4. Мікросервісна архітектура

У мікросервісній моделі вебдодаток складається з низки незалежних сервісів, кожен з яких реалізує окрему функцію. Цей підхід ідеально підходить для масштабованих систем, наприклад, Amazon, Netflix, але потребує складнішого налаштування взаємодії між сервісами, зазвичай через API.

5. Serverless-архітектура, цей підхід полягає у використанні хмарних функцій для виконання логіки, без розгортання традиційного серверного додатку. Підходить для невеликих або окремих модулів вебсайтів, які повинні швидко масштабуватись.

Зважаючи на обрану специфіку проекту — створення повноцінного вебзастосунку інтернет-магазину з інтерактивним інтерфейсом, було обрано компонентну модель проектування на базі бібліотеки React. Ця модель дозволяє:

- розділити функціональність на незалежні компоненти, такі як сторінки, шапка, футе, форма пошуку, картка товару тощо;
- повторно використовувати код, наприклад, один компонент товару — на головній, у колекції та у кошику;
- краще контролювати стан додатку, через React hooks, Context API або сторонні бібліотеки;
- забезпечити високу гнучкість і масштабованість застосунку.

Підсумовуючи, такий підхід найкраще підходить для динамічних сайтів із багатьма сторінками, як-от інтернет-магазини.

2.4. Вибір інструментів для розробки вебсайту

Як сказано вище, інструменти створення вебдодатків є доволі важливими, оскільки використовуючи їх можливо цілком легко створити повноцінний продукт, по типу різноманітних вебдодатків, які можуть бути відкриті в Інтернеті і використовуватися користувачами з різних місць світу.

Крім того використання саме правильних інструментів може допомогти зробити процес створення вебдодатків більш ефективним та продуктивним.

Тому для розробки веб-додатку інтернет магазину буде застосовувано саме такі інструменти:

1. Саме основною мовою програмування буде JavaScript, яка представляє з себе мову програмування, яка широко використовується для створення динамічних, інтерактивних веб-сайтів та веб-додатків. Також, це мова високого рівня, що означає, що вона розроблена так, щоб її було легко читати і писати, і часто використовується разом з HTML і CSS для додавання інтерактивності та функціональності веб-сторінкам. Є окремий сенс його застосовувати для створення веб-додатку [28].

2. Створення фронтенд частини сайту буде здійснено за допомогою React – це JavaScript-бібліотека для створення інтерфейсів користувача. Це популярний інструмент для створення веб-додатків з високою динамічністю, що дозволяє швидко та ефективно створювати складні веб-додатки. React був розроблений Facebook, а потім був переданий в організацію ReactJS Foundation. Ці інструменти дозволяють створити веб-сайт зі статичним контентом, який можна переглянути в браузері [29].

Для того щоб інсталювати його, достатньо просто зайти на офіційний сайт розробників React

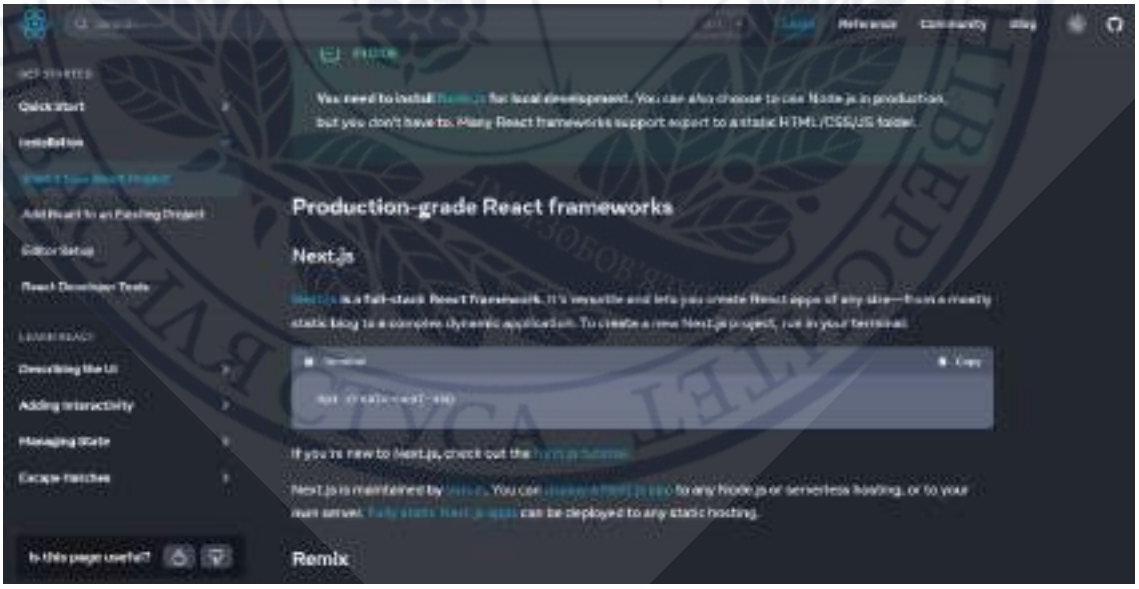

Рисунок 2.1 – Інсталяція реакту

І ввести в термінал команду npx create-next-app, яка є інструментом для створення нового проекту на основі фреймворку Next.js. Вона використовується для швидкого старту проекту, а саме:

1. Створення нової папки з назвою проекту.

2. Ініціалізація Git репозиторію для контролю версій.

3. Інсталяція необхідних пакетів для розробки проекту, таких як Next.js, React і ReactDOM.

4. Запуск початкового сервера Next.js.

Ця команда дозволяє розробникам швидко створювати нові проекти з використанням фреймворку Next.js і починати працювати з ними без необхідності налаштовування оточення розробки.

3. Для створення бекенд частини, буде застосовано Node.js, який відносно загального визначення представляє з себе серверну платформу, яка дозволяє розробникам використовувати JavaScript для створення серверних застосунків. Він використовує подвійний цикл, щоб забезпечити неблокуючу та ефективну обробку запитів, а також надає доступ до більшості стандартних модулів, таких як HTTP, для обробки запитів [30].

Однак для роботи з ним, потрібно його спочатку встановити, а для цього потрібно скачати останню його версію з офіційного сайту.

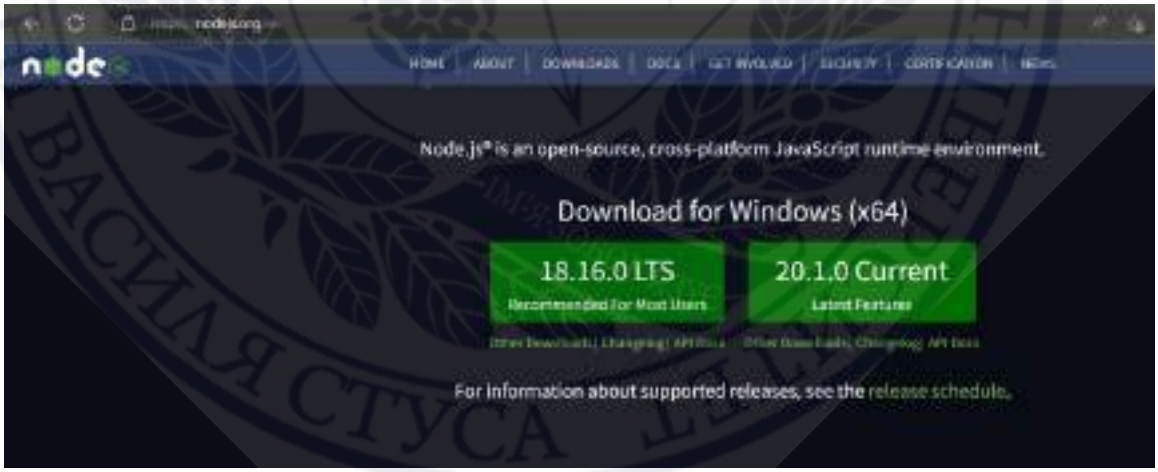

Рисунок 2.2 – Встановлення Node.js

Для досягнення оптимальних результатів у розробці серверної частини сайту, буде обрано бібліотеку Express.js, яка є одним із найпопулярніших фреймворків для створення веб-застосунків на платформі Node.js. Express.js відомий своєю легкістю, гнучкістю та високою ефективністю, що дозволяє розробникам створювати серверні застосунки швидко та з мінімальними

витратами часу на налаштування. Це дозволяє зосередитись безпосередньо на розробці функціональності, що є основним перевагою для багатьох проектів.

Однією з основних переваг Express.js є його здатність обробляти запити та взаємодіяти з клієнтами через різні типи маршрутизації [31]. Він дозволяє створювати прості та зручні маршрути для обробки запитів різних типів, таких як GET, POST, PUT, DELETE. Кожен маршрут в Express відповідає на певний HTTP-запит, що дає змогу ефективно обробляти інформацію, отриману від користувачів, а також передавати дані на клієнтську частину.

4. Для створення ефективного API та обміну даними між сервером та клієнтом, буде використано бібліотеку Axios. Axios є популярним інструментом для виконання HTTP-запитів із клієнтської частини додатку. Ця бібліотека дозволяє відправляти запити різних типів, таких як GET, POST, PUT та DELETE, а також отримувати відповіді у різних форматах, таких як JSON, XML, CSV тощо. Однією з основних переваг Axios є його простота в інтеграції з React, оскільки бібліотека підтримує проміси, що дає змогу організувати асинхронні запити без складної обробки зворотного виклику [32].

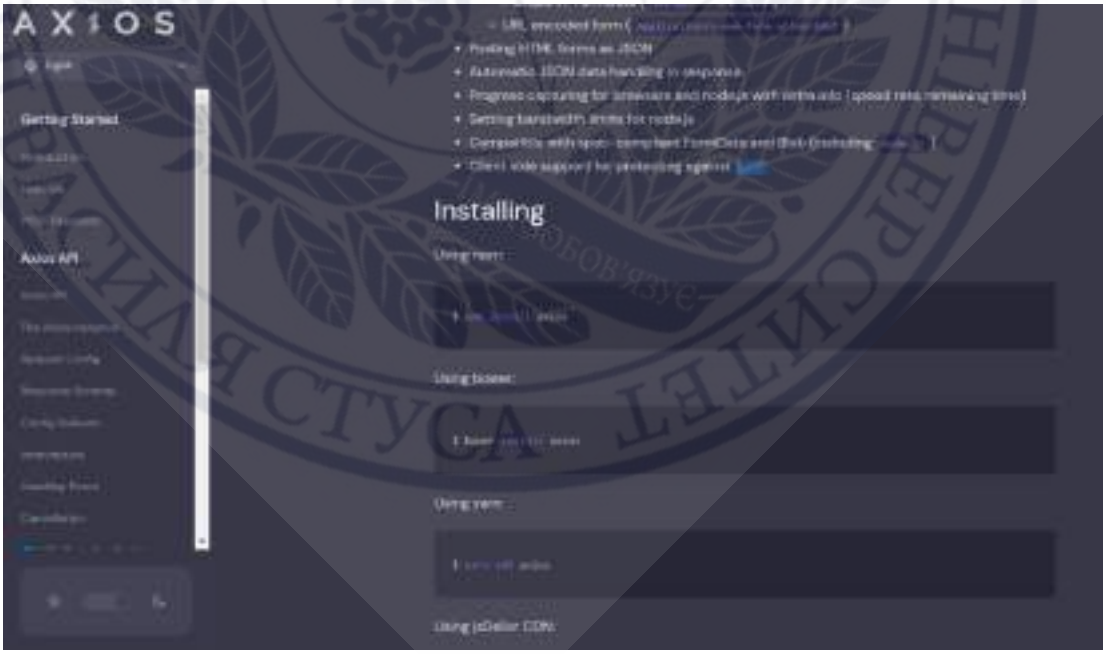

Рисунок 2.3 – Встановлення бібліотеки Axios

Поєднання Express.js та Axios дозволяє створювати потужну систему для обробки запитів та відповіді на них. За допомогою Express.js на сервері можна

створювати маршрути для обробки запитів, що надходять з клієнтської частини. Наприклад, коли користувач ініціює запит через інтерфейс вебсайту, Axios відправляє запит до серверу, де Express.js обробляє його, звертається до бази даних, виконує необхідні операції та повертає результат у вигляді JSON-об’єкта.

У процесі розробки мого сайту, Axios буде використаний для здійснення запитів до API, яке обробляється через Express.js на серверній стороні. Це дозволить зручно та швидко взаємодіяти з даними, оптимізувати взаємодію між клієнтською та серверною частинами та зробити весь процес розробки максимально ефективним.

5. Для розробки веб-додатків використовують редактори коду, такі як Visual Studio Code, Atom, Web Storm або Sublime Text. Ці інструменти дозволяють розробникам швидко та легко редагувати та зберігати код. Але для розробки буде використано саме visual studio code [33].

6. Для створення веб-додатків на React важливо вибрати відповідний інструмент для збірки та оптимізації коду. Одним з таких інструментів є Vite. Це сучасний збирач проектів, який значно покращує процес розробки завдяки швидкому завантаженню та ефективному управлінню залежностями. Обрано Vite, оскільки він забезпечує миттєву перезагрузку сторінок та чудову продуктивність завдяки використанню сучасних стандартів, таких як ES-модулі [34]. Окрім того, Vite має просту конфігурацію і чудово працює з React, що дозволяє зосередитись на розробці інтерфейсу без необхідності налаштовувати складні параметри збірки.

7. Для використання систем контролю версій чи простіше кажучи для збереження та управління кодом використовують системи контролю версій, застосовують або Git, або SVN. Ці інструменти дозволяють розробникам зберігати та відстежувати зміни у коді, а також працювати в команді над проектом [25].

8. Також, Бази даних є важливою складовою бакену веб-додатків, які забезпечують зберігання, організацію та доступ до даних. Тому для зберігання даних, було обрано NoSQL базу даних замість SQL, оскільки проект буде

потребувати роботи з великим обсягом даних та швидкої їх обробки. Єдиними таблицями SQL-баз даних, як правило, є реляційні таблиці, що може зробити роботу зі складними, нереляційними даними більш витратною. Крім того, NoSQL бази даних, які використовують документ-орієнтований підхід, зазвичай можуть бути більш масштабованими та еластичними, що дозволяє легко додавати нові дані без втрати продуктивності бази даних. А з Nosql, було обрано MongoDB, оскільки вона дозволяє зберігати структуровані та неструктуровані дані в документах. Це дає можливість зберігати дані у вигляді об'єктів JSON, що дозволяє легко зберігати та отримувати дані без необхідності змінювати схему бази даних [35].

9. Для виведення повідомлень про помилки, буде застосовано бібліотеку tostify, яка зі сторони дизайну зовнішньо непогано показує повідомлення про помилки.

10. Для тестування API-запитів у проекті обрано Thunder Client — зручний плагін для Visual Studio Code. Це інструмент, який дозволяє швидко і ефективно перевіряти серверні запити без потреби переходити до сторонніх додатків, таких як Postman. Thunder Client підтримує основні HTTP-методи GET, POST, PUT, DELETE і дозволяє налаштовувати запити з параметрами, заголовками та тілом запиту. Обрано Thunder Client, оскільки він інтегрується безпосередньо у редактор коду, що зручно та прискорює процес розробки і тестування API під час роботи над проектом.

11. Для стилізації веб-додатків одним із найпопулярніших інструментів є Tailwind CSS. Це утилітний CSS-фреймворк, який дозволяє швидко створювати кастомізовані дизайни без написання великої кількості CSS-коду. Обрано Tailwind CSS для проекту, оскільки він забезпечує гнучкість і зручність у розробці інтерфейсів завдяки системі класів для прямого застосування стилів до елементів HTML. Цей підхід дозволяє швидко та ефективно змінювати зовнішній вигляд елементів без необхідності створення окремих CSS-файлів. Tailwind також добре інтегрується з React, дозволяючи підтримувати чистоту коду та зменшувати кількість зайвих стилів.

12. Для маршрутизації на фронтенд частині буде використано React Router DOM – це бібліотека React, яка дозволяє легко керувати маршрутизацією на клієнтській стороні вебдодатка. З його допомогою можна створити компоненти-маршрутизатори, які забезпечують відображення відповідних компонентів, залежно від шляху URL [36].

Загалом, використання інструментів для розробки веб-додатків може допомогти розробникам зекономити час та зусилля при створенні вебдодатків, а також зробити їх більш ефективними та продуктивними.

РОЗДІЛ 3
РОЗРОБКА БЕКЕНД ТА ФРОНТЕНД ЧАСТИН

3.1. Структура бази даних

У контексті реалізації інтернет-магазину структурна схема бази даних включає кілька основних таблиць, зокрема:

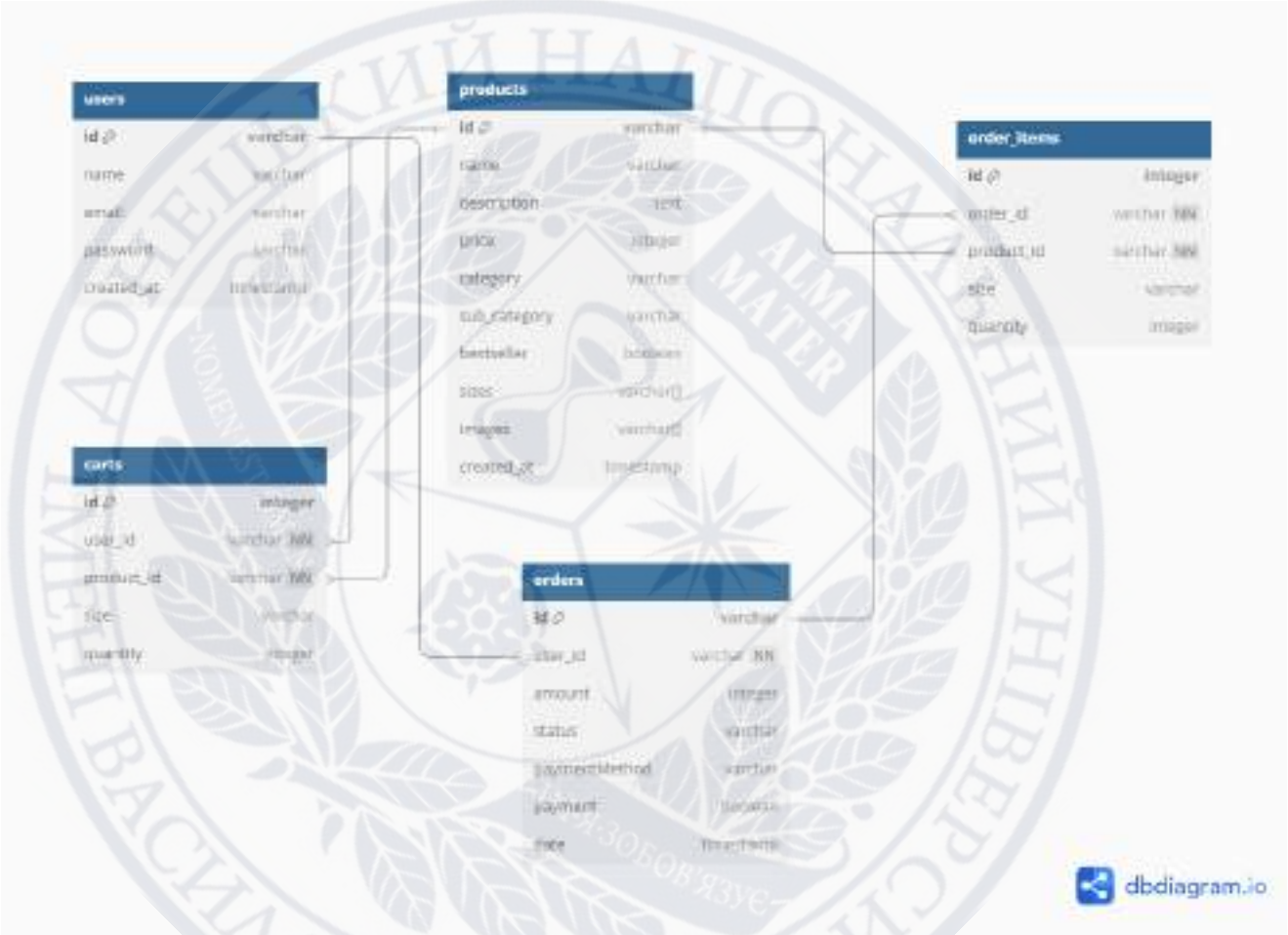

Рисунок 3.1 – Ілюстрація структури бази даних

Таким чином, що користувач може мати багато замовлень та корзин з речами, в замовленні може бути багато товарів різної кількості і розмірів. Товар в замовленнях може бути багато разів різної кількості і розмірів, так само які в корзині.

В результаті створено такі таблиці бази даних як: таблиці товару, користувача та замовлень.

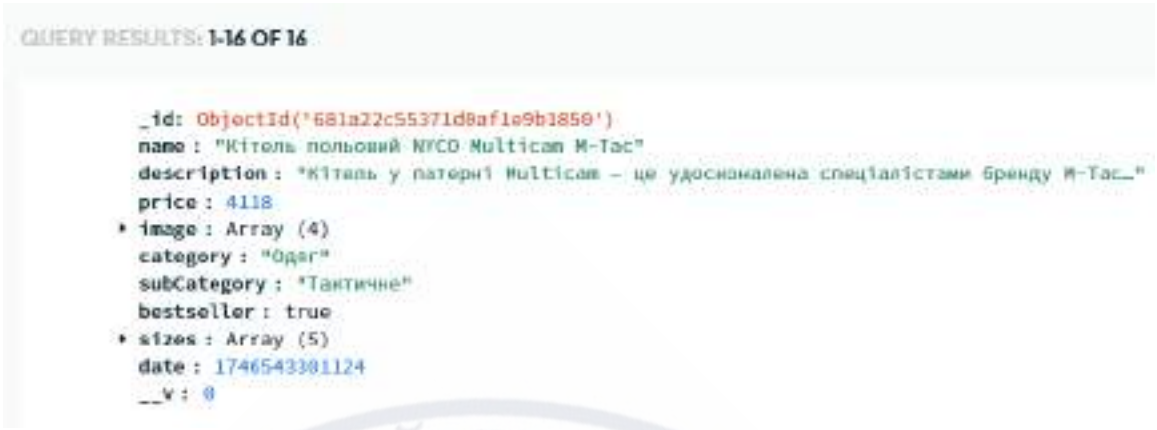

Рисунок 3.2 – Таблиця товару

Опис полів таблиці:

- name — ім’я товару;
- description — опис товару;
- price — ціна товару;
- image — посилання на фото товару;
- category — категорія товару;
- subCategory – підкатегорія товару
- bestseller – чи є товар популярним;
- sizes – розміри товару;
- date – дата додавання;
- __v – службове поле для відстеження версії документа;

Рисунок 3.3 – Таблиця користувача

Опис полів таблиці:

- name — ім’я користувача;
- email — електронна пошта;
- password —хешований пароль користувача;

- isAdmin — логічне значення, що вказує, чи є користувач адміністратором системи;
- cartData — масив з товарами в корзині користуваача;
- __v – службове поле для відстеження версії документа.

Рисунок 3.4 – Таблиця замовлення

Опис полів таблиці:

- _id – унікальний ідентифікатор замовлення, який створює MongoDB;
- userId – ідентифікатор користувача, що зробив замовлення;
- items – масив товарів, які входять до замовлення;
- amount – загальна сума замовлення;
- address – об’єкт з інформацією про адресу доставки;
- status – поточний статус замовлення;
- paymentMethod – спосіб оплати, обраний користувачем;
- payment – ознака, чи було замовлення оплачено;
- date – дата створення замовлення у мілісекундах;
- __v – службове поле для відстеження версії документа.

3.2. Структура сайту

Структурна схема вебзастосунку інтернет-магазину, відображає логічну побудову основних сторінок, підрозділів і шляхів навігації між ними. Такий

підхід дозволяє не лише забезпечити зручність користувача, а й реалізувати гнучке масштабування проєкту в майбутньому.

Рисунок 3.5 – Ілюстрація структури сайту

Головна сторінка є центральним вузлом застосунку, з якого користувач має доступ до всіх основних розділів. Вона містить блоки популярних товарів, рекомендацій, переходів до каталогу, інформаційних сторінок.

Популярні товари та каталог, ці два розділи мають схожу внутрішню структуру. Кожен містить:

- сторінку товару, де представлено детальну інформацію, фото, ціну, доступні розміри тощо;
- рекомендовані товари, які формуються на основі вибраного товару або історії переглядів;
- зворотний зв’язок, що дозволяє користувачеві надіслати питання або зауваження.

Інформаційні сторінки містять:

- про нас – містить загальні відомості про компанію, її місію та діяльність;
- контакт – надає інформацію для зв’язку та доступ до форми зворотного зв’язку.

Меню користувача передбачає:

- вхід користувача – реалізований через JWT-автентифікацію;
- замовлення – дає змогу переглядати історію покупок;

- вихід з акаунту – виконується через очищення токена з localStorage.

Розділ кошика дозволяє керувати замовленнями. Передбачено кілька етапів:

- кошик – перегляд обраних товарів;
- оплата – перевірка замовлення та вибір методу оплати;
- оплата через Stripe – інтеграція з платіжною системою Stripe для безпечних транзакцій;
- замовлення – підтвердження оформлення покупки.

Рисунок 3.6 – Ілюстрація структури панелі адміністратора

Структура панелі адміністратора має в собі такі блоки:

- вхід користувача – початковий екран для авторизації адміністратора; доступ до панелі адміністратора відкривається лише після успішного входу, як правило, через JWT або сесію;
- головна сторінка панелі адміністратора – центральна точка керування, з якої можна переходити до трьох основних функціональних блоків;
- додати товари – інтерфейс для створення нових товарів: введення назви, опису, ціни, категорії, зображень та розмірів; передбачена валідація даних;
- список товарів – перелік усіх товарів з можливістю редагування, видалення або фільтрації за параметрами, наприклад, категорія, наявність, популярність;
- список замовлень — відображення всіх замовлень, зроблених

3.3. Розробка бекенд частини вебсайту

Розробка API на Node.js полягає в створенні веб-додатку, який надає програмні інтерфейси (API) для інших програм або систем. В API можна використовувати різні методи HTTP, такі як GET, POST, PUT та DELETE для звернення до даних або виконання операцій.

Для розробки API зі сторони бекенд частини як було описано вище, застосовуватиметься фреймворк Express.js, що дозволяє швидко створювати веб-додатки з маршрутизацією та можливими маршрутами і зв’язком з моделями [31].

Таким чином попередньо створивши відповідну модель Product бази даних Mongodb, можливо налаштувати маршрути.

Рисунок 3.7 – Налаштування маршрутів

Маршрут за шляхом /add обробляє POST-запит. Він дозволяє адміністраторам додавати нові товари до бази даних. Перед виконанням основної логіки додається перевірка прав доступу adminAuth та обробка зображень, які прикріплюються до товару, за допомогою upload.fields(). Основна логіка додавання реалізована у функції addProduct, яка визначена у відповідному контролері.

Маршрут за шляхом /remove відповідає за видалення товару з бази даних і доступний лише авторизованим адміністраторам. Запит має містити

ідентифікатор товару, який необхідно видалити, а обробку виконує функція removeProduct.

Маршрут за шляхом /single реалізує можливість отримання повної інформації про один конкретний товар на основі його ідентифікатора, переданого у тілі запиту. Запит обробляється функцією singleProduct.

Маршрут за шляхом /list відповідає за повернення повного списку товарів, що зберігаються в базі даних. Це відкритий GET-запит, який не вимагає авторизації, і результат формується за допомогою функції listProduct.

Таким чином, описана система маршрутизації дозволяє повністю управляти товарами на серверній стороні — додавати, видаляти, переглядати список або отримувати конкретний товар, при цьому дотримуючись правил авторизації та обробки файлів.

Таким чином, описана система маршрутизації дозволяє повністю управляти товарами на серверній стороні — додавати, видаляти, переглядати список або отримувати конкретний товар, при цьому дотримуючись правил авторизації та обробки файлів.

А зі сторони фронтенду буде використано фреймворк Axios для створення http запитів, щоб отримати дані з фронтенд частини сайту зі сторони клієнту.

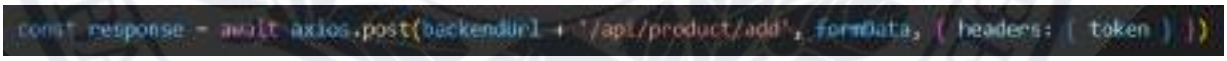

Рисунок 3.8 – Застосування Axios в коді

Цей рядок виконує POST-запит за вказаною адресою (/api/product/add), відправляючи зібрані дані formData та заголовок з токеном авторизації. Токен необхідний для перевірки прав доступу користувача [20].

Далі обробляється відповідь сервера:

- якщо запит успішний і сервер повертає success: true, користувач отримує повідомлення про успішне додавання, а поля форми очищуються;
- у разі помилки або невдалої відповіді, користувачу показується повідомлення з описом помилки.

Таким чином, axios в даному прикладі забезпечує зручну та безпечну взаємодію з бекендом, дозволяючи додавати товари з фронтенду до бази даних.

Крім того, для захисту паролів користувачів можна використовувати бібліотеку bcrypts.js, яка дозволяє зберігати хеші паролів замість самого пароля. Такий підхід зменшує ризик витоку паролів в разі злому сайту або бази даних, оскільки навіть якщо зловмисник здобуде доступ до збережених хешів, він не зможе одержати самі паролі[19].

Для забезпечення автентифікації на сервері створено три окремі маршрути: для входу користувача, реєстрації користувача та входу адміністратора. Усі вони реалізовані з використанням бібліотек bcryptjs для хешування паролів та jsonwebtoken для створення JWT-токенів.

Під час реєстрації дані, надіслані з клієнта (name, email, password), спочатку перевіряються на коректність. Якщо вказаний email уже існує в базі даних або має некоректний формат, сервер повертає відповідне повідомлення про помилку. Також перевіряється складність пароля (мінімум 8 символів).

Після проходження валідації пароль хешується з використанням bcryptjs, а новий користувач зберігається в базі даних. Успішна реєстрація завершується створенням JWT-токена, який повертається клієнту у відповіді:

```
const salt = await bcrypt.genSalt(10)
const hashedPassword = await bcrypt.hash(password, salt)
const newUser = new userModel({
    name,
    email,
    password: hashedPassword
})
const user = await newUser.save()
const token = createToken(user._id)
res.json({success: true, token})
```

Рисунок 3.9 – Створення JWT токену

Під час логіну сервер шукає користувача за поштою, а потім порівнює введений пароль із хешованим паролем у базі даних за допомогою bcrypt.compare(). Якщо автентифікація успішна, створюється токен, якщо ні – виводиться помилка.

```
const isMatch = await bcrypt.compare(password, user.password)

if(isMatch){
    const token = createToken(user._id)
    res.json({success: true, token})
}
else{
    res.json({success: false, msg: "Неправильні облікові дані"})
}
```

Рисунок 3.10 – Перевірка даних для автентифікації

У разі помилки користувач отримає повідомлення про некоректні облікові дані або відсутність облікового запису.

Окремо реалізований маршрут входу для адміністратора. У цьому випадку перевіряються вхідні дані на відповідність збереженим у .env файлі обліковим даним адміністратора.

```
if(email === process.env.ADMIN_EMAIL && password === process.env.ADMIN_PASSWORD){
    const token = jwt.sign(email+password, process.env.JWT_SECRET)
    res.json({success: true, token})
}
```

Рисунок 3.11 – Перевірка вхідних даних для адміністратора

Успішний вхід також завершується видачею токена, що дозволяє адміністратору отримати доступ до обмежених частин сайту.

Цей підхід до авторизації дозволяє покращити безпеку сайту, оскільки пароль користувача не зберігається відкритим текстом в базі даних, а замість цього зберігається його хеш. Крім того, використання JWT-токенів дозволяє підтвердити ідентичність користувача на кожному запиті до сервера, забезпечуючи тим самим додатковий рівень безпеки.

Загалом, реалізована система автентифікації поєднує в собі сучасні засоби безпеки: хешування паролів, перевірку даних користувача та авторизацію за допомогою JWT, що забезпечує захист даних користувачів і контроль доступу на вебсайті.

Для збереження та структурування даних про товари в базі даних використовується модель productModel, створена за допомогою фреймворку Mongoose.

Рисунок 3.12 – Модель продукту

У моделі визначені основні характеристики товару:

- name – назва товару;
- description – короткий опис товару;
- price – вартість;
- image – масив зображень, наприклад, для галереї;
- category та subCategory – категорія та підкатегорія, до якої належить товар;
- bestseller – булеве значення, яке вказує, чи є товар популярним;
- sizes – список доступних розмірів, опційно;
- date – дата додавання товару у вигляді числа, наприклад, timestamp.

Ця модель дозволяє зручно працювати з товарами в базі даних та реалізовувати функціонал інтернет-магазину.

Нарешті, за допомогою методу mongoose.model() створюється модель Product, яка буде використовуватися для взаємодії з колекцією продуктів у базі даних.

Далі за допомогою цього ж фреймворку, таким чином підключено сервер до бази даних, застосовуючи окрему функцію підключення.

Рисунок 3.13 – Підключення mongoose

Де перш за все з прикладу, ми імпортуємо бібліотеку mongoose та dotenv.

Бібліотека dotenv дозволяє зберігати конфігураційні дані у файлі .env, що забезпечує більш безпечне зберігання конфігурації.

Далі створюється асинхронна функція connectDB(), в якій визначено обробник події connected. Коли з'єднання з базою даних буде встановлено, в консоль виводиться повідомлення "DB connected".

Після цього викликається метод mongoose.connect(), який здійснює підключення до бази даних. Адреса з'єднання передається у вигляді змінної середовища process.env.MONGODB_URL, до якої додається назва бази (/e-commerce). Такий підхід дозволяє гнучко змінювати конфігурацію без потреби змінювати код, адже дані зчитуються з файлу .env.

Функція connectDB експортується і надалі використовується в головному серверному файлі для ініціалізації підключення при запуску додатку.

Для забезпечення безпеки при реєстрації та вході користувачів на вебсайті, використовуються сучасні методи аутентифікації, які включають в себе хешування паролів за допомогою бібліотеки bcrypt та генерацію токенів JWT для підтвердження особи користувача. Далі буде надано детальне пояснення функцій, що реалізують ці процеси, зокрема — для входу, реєстрації користувача та адміністраторського доступу. Загалом цей процес працює за таким алгоритмом:

1. Створення токену для користувача

В основі процесу аутентифікації лежить створення токену, що містить ідентифікатор користувача. Для цього використовуються JSON Web Token (JWT), що є надійним засобом для безпечного зберігання і передачі інформації між клієнтом і сервером [37]. Токен створюється на основі унікального ID користувача, після чого він може бути використаний для подальших перевірок аутентифікації.

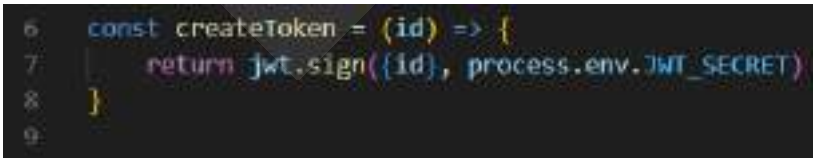

Рисунок 3.14 – Створення токену

У функції createToken токен генерується через використання бібліотеки jsonwebtoken, де передається id користувача, а також секретний ключ, що зберігається у змінній середовища JWT_SECRET. Це дозволяє забезпечити безпечну підписку на токен.

2. Вхід користувача

Функція loginUser відповідає за аутентифікацію користувача. Під час процесу входу сервер перевіряє наявність користувача в базі даних за допомогою введеної електронної пошти. Якщо користувач існує, здійснюється порівняння введеного пароля з тим, що зберігається в базі даних. Для цього використовується метод bcrypt.compare(), який дозволяє безпечно порівнювати хешовані паролі.

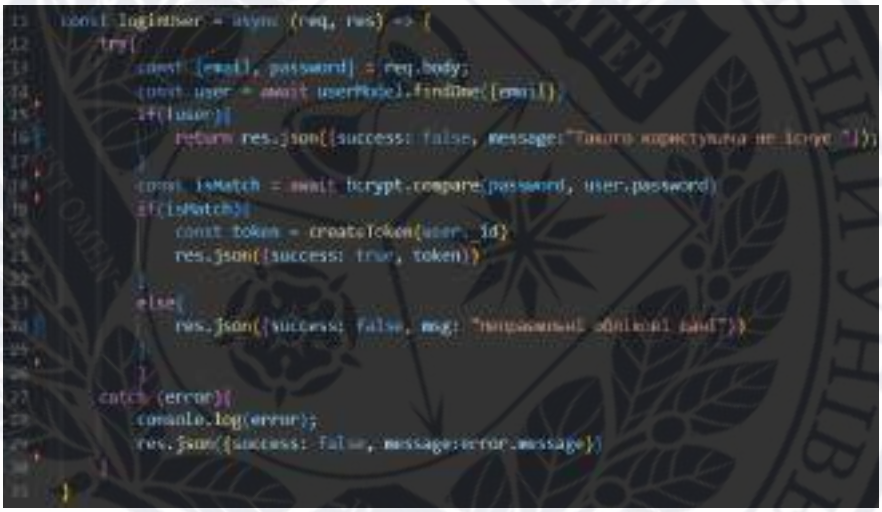

Рисунок 3.15 – Функція loginUser

Якщо введений пароль збігається з хешованим у базі даних, генерується JWT токен, що передається користувачу для подальшого використання.

3. Реєстрація користувача

У процесі реєстрації користувачів важливо не тільки перевіряти унікальність електронної пошти, а й забезпечувати дотримання правил безпеки для введеного пароля. Для цього застосовуються валідації за допомогою бібліотеки validator для перевірки формату електронної пошти, а також перевірка мінімальної довжини пароля.

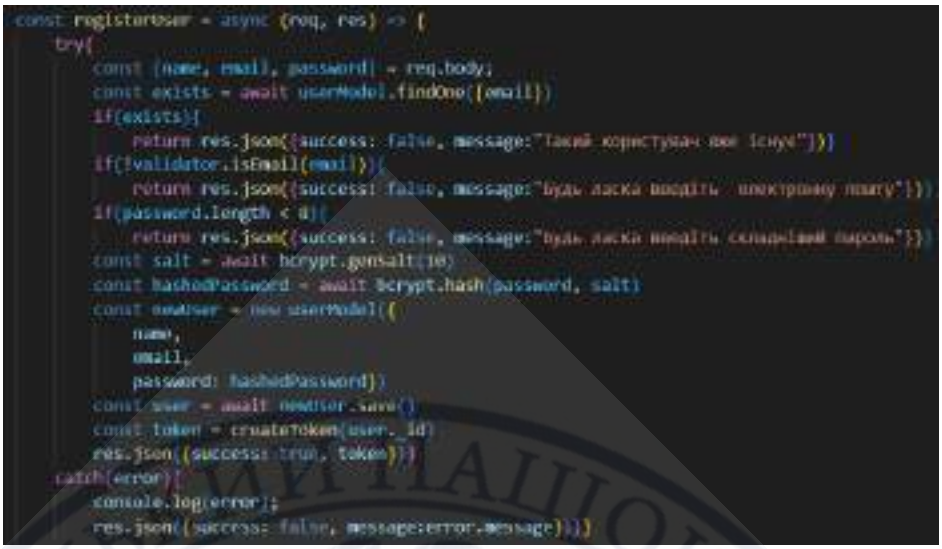

Рисунок 3.16 – Функція реєстрації користувачів

У разі успішної реєстрації, після хешування пароля, створюється новий запис у базі даних, і користувачу видається токен для подальшого доступу.

4. Логін адміністратора

Для доступу до адміністративної частини сайту передбачена окрема функція adminLogin, яка дозволяє увійти адміністратору, перевіряючи спеціальні облікові дані для адміністратора. В разі правильних введених даних, електронної пошти та пароля, генерується JWT токен, що надається адміністратору для доступу до захищених ресурсів.

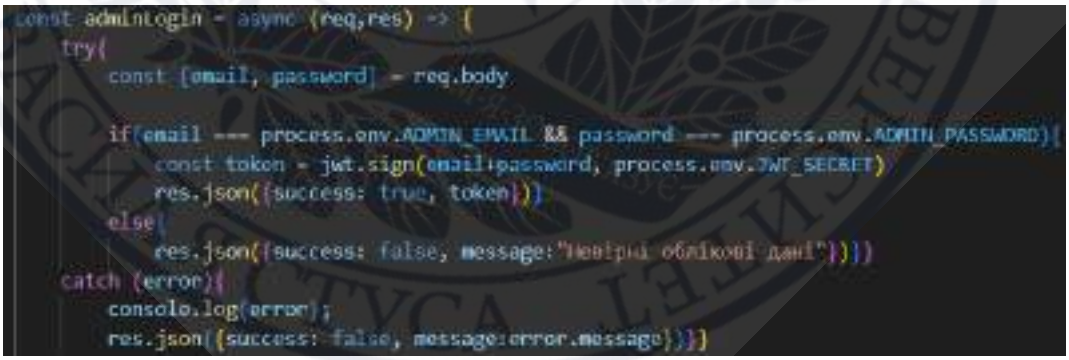

Рисунок 3.17 – Функція adminLogin

Таким чином, у даній реалізації забезпечено комплексний підхід до аутентифікації користувачів за допомогою хешування паролів та токенів JWT. Цей підхід дозволяє зберігати високий рівень безпеки, уникати зберігання паролів у відкритому вигляді та забезпечувати контроль доступу до різних частин вебсайту через перевірку автентичності користувачів та адміністраторів.

Весь основний код з бекендом та фронтендом програми розміщено в додатку, де представлено всі функції та елементи, які використовуються для створення бекенд частини сайту а також його фронтенд частини.

Додаток включає код, що забезпечує роботу сервера, маршрутизаторів і всіх інших частин коду, які приводять в дію весь сайт в додатку Б.

3.4. Створення фронтенд частини вебсайту

Основна навігація для користувача реалізована у головному компоненті App, де за допомогою бібліотеки react-router-dom здійснюється маршрутизація між сторінками.

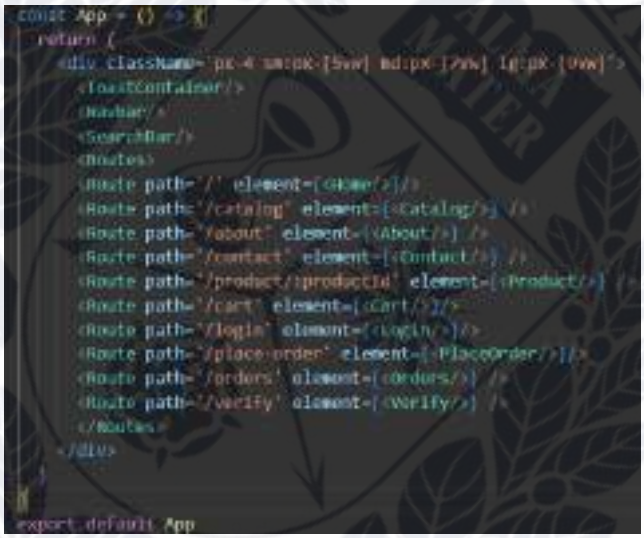

Рисунок 3.18 – Навігація вебсайту

У даному прикладі спочатку підключаються всі необхідні компоненти, які представляють окремі сторінки сайту: головна, каталог, сторінка товару, сторінка з кошиком тощо. Компонент Navbar відображається на всіх сторінках для зручної навігації, а SearchBar — для швидкого пошуку товарів. Повідомлення про події (наприклад, про додавання до кошика або помилки) виводяться за допомогою бібліотеки react-toastify та компонента ToastContainer. Кожен шлях містить шляховий шаблон та компонент, який буде рендеритися, коли користувач перейде на цей шлях [38].

Окремо реалізована адміністративна частина сайту, яка також базується на бібліотеці react-router-dom. Основний файл App у цьому випадку має трохи іншу

логіку, оскільки включає перевірку токена авторизації адміністратора.

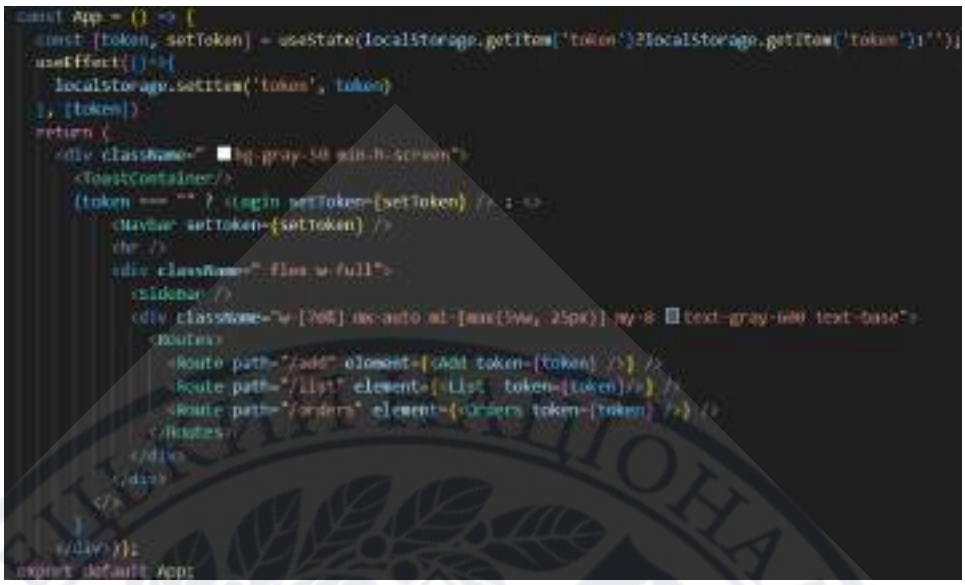

Рисунок 3.19 – Головна сторінка адміністратора

У цьому коді спочатку з локального сховища отримується токен доступу. Якщо токен відсутній, користувач бачить компонент Login. Якщо токен існує, він має доступ до адміністративних сторінок:

- /add — сторінка додавання товарів;
- /list — перегляд/редагування товарів;
- /orders — перегляд замовлень.

Також для навігації використовується компонент SideBar, який розташований ліворуч і дає змогу швидко переходити між адміністративними розділами.

Таким чином, реалізація маршрутизації в адміністративній частині також гнучка й масштабована, з врахуванням авторизації через токен.

У процесі розробки вебзастосунку важливим етапом є створення окремих сторінок, які відповідають за відображення основного контенту. Однією з таких є головна сторінка, на якій відображаються популярні товари та форма підписки на новини. Для її реалізації застосовано компонентний підхід із використанням функціональних компонентів React [39].

Нижче наведено приклад сторінка Home, який виконує роль простої сторінки та імпортує два підкомпоненти: BestSeller — для виводу найпопулярніших товарів, та NewsLetterBox — для блоку підписки на новини.

Рисунок 3.20 – Сторінка Home

Компонент BestSeller реалізує логіку вибору та відображення хітів продажу серед доступних товарів. Джерелом даних у цьому випадку виступає глобальний контекст — ShopContext, який забезпечує доступ до стану застосунку з будь-якого компонента, без необхідності передавання даних через пропси.

Рисунок 3.21 – Компонент BestSeller

На початку компонента за допомогою хука useContext здійснюється доступ до глобального масиву products. Далі, за допомогою useEffect, фільтруються товари, що мають позначку bestseller, після чого формується підмножина з п’яти перших елементів, яка зберігається в локальному стані bestSeller.

Цей підхід дозволяє централізовано зберігати дані про товари, зменшуючи

кількість дубльованих запитів до серверу, покращуючи продуктивність і спрощуючи управління станом. Таким чином, реалізація простої сторінки досягається шляхом композиції повторно використовуваних компонентів та логіки, що відповідає сучасним підходам до побудови Single Page Application вебсайтів [40].

У процесі розробки інтернет-магазину ключовим завданням стало забезпечення ефективного керування глобальним станом додатка. Це включає в себе управління товарами, кошиком, токеном автентифікації користувача, функціональністю пошуку, а також взаємодію з бекендом через REST API. Для реалізації цієї мети було створено глобальний контекст ShopContext, що ґрунтується на бібліотеках React, axios, react-toastify та react-router-dom.

Код реалізує шаблон контекстного провайдера під назвою ShopContextProvider, який виступає джерелом глобального стану. Усі дочірні компоненти, що обгорнуті цим провайдером, мають доступ до значень і функцій, переданих через контекст.

1. Змінні useState охоплюють усі критично важливі аспекти: товари, токен автентифікації, дані кошика, пошук та інші.

```
const currency = "2";
const delivery_fee = 10;
const backendUrl = import.meta.env.VITE_BACKEND_URL
const [search, setSearch] = useState("");
const [showSearch, setShowSearch] = useState(false);
const [cartItems, setCartItems] = useState({});
const navigate = useNavigate()
const [token, setToken] = useState('')
const [products, setProducts] = useState([])
```

Рисунок 3.22 – Використання useState

2. Через бібліотеку axios реалізовано запити до серверу /api/product/list, також /api/cart/add, /update, /get для взаємодії з кошиком. Всі запити супроводжуються перевіркою token.

```
const addToCart = async (itemId, size, sizes = []) => {
  if (sizes.length > 0 && !size) {
    toast.error("Оберіть розмір товару");
    return;
  }
  let cartData = structuredClone(cartItems);
  if (cartData[itemId]) {
    if (cartData[itemId][size]) {
      cartData[itemId][size] += 1;
    } else {
      cartData[itemId][size] = 1
    }
  } else {
    cartData[itemId] = {};
    cartData[itemId][size] = 1
  }
  setCartItems(cartData);
  if (token) {
    try {
      await axios.post(backendUrl + '/api/cart/add', { itemId, size }, { headers: { token } });
    } catch (error) {
      console.log(error);
      toast.error(error.message)}}
```

Рисунок 3.23 – Функція addToCart

```
const updateQuantity = async (itemId, size, quantity) => {
  let cartData = structuredClone(cartItems);
  cartData[itemId][size] = quantity;
  setCartItems(cartData);
  if (token) {
    try {
      await axios.post(backendUrl + '/api/cart/update', {itemId, size, quantity}, {headers:{token}})
    } catch (error) {
      console.log(error);
      toast.error(error.message)}};
```

Рисунок 3.24 – Функція updateQuantity

3. Один з useEffect слідкує за наявністю токена в localStorage, автоматично витягуючи його після перезавантаження сторінки.

```
useEffect(()=>{
  if (!token && localStorage.getItem('token')) {
    setToken(localStorage.getItem('token'))
    getUserCart(localStorage.getItem('token'))
  }
})
```

Рисунок 3.25 – Застосування useEffect

4. Бібліотека react-toastify використовується для інформування користувача про помилки.

```
const addToCart = async (itemId, size, sizes = []) => {
  if (sizes.length > 0 && !size) {
    toast.error("Оберіть розмір товару");
    return;
  }
```

Рисунок 3.26 – Застосування react-toastify

5. В усіх місцях, де можливі виключення, реалізовано обробку помилок. Це дозволяє запобігти збоям.

```
if (token) {
  try {
    await axios.post(backendUrl + '/api/cart/update', {itemId, size, quantity}, {headers:{token}})
  } catch (error) {
    console.log(error);
    toast.error(error.message)}}
```

Рисунок 3.27 – Обробка помилок

Загалом, цей фрагмент реалізує ефективну модель управління станом, дотримуючись принципів компонентності, інкапсуляції логіки та реактивності. React Context API забезпечує зручний і гнучкий доступ до спільного стану.

3.5. Спосіб запуску та демонстрація створеного вебсайту

Спочатку потрібно:

1. Зайти в кореневу папку з бекендом, за допомогою команди cd backend, а потім прописати в терміналі “npm run server”, щоб запустити роботу серверу та мати доступ до даних з бази даних.

2. Далі потрібно зайти папку з фронтендом, за допомогою команди cd frontend, потім прописати в терміналі “npm run dev”, щоб запустити роботу сайту.

Після цього нам відкриється таке вікно з головною сторінкою сайту:

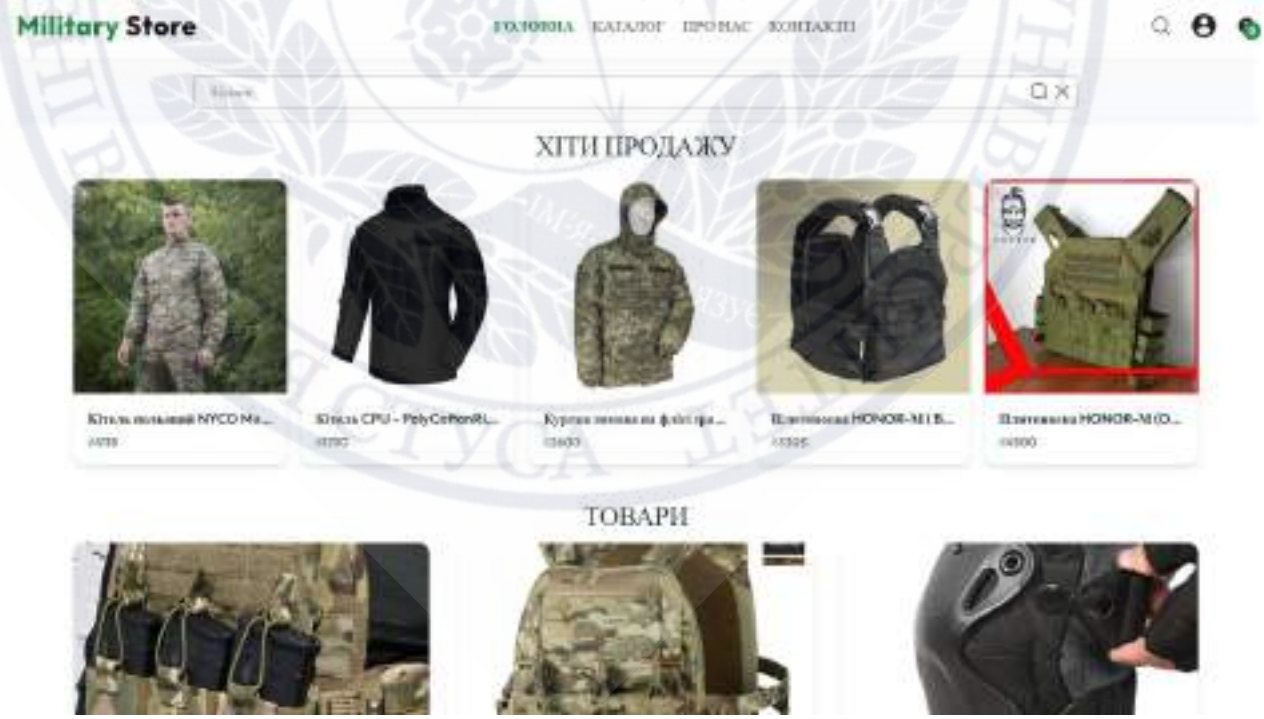

Рисунок 3.28 – Головна сторінка сайту

Демонстрація сайту див. в додатку А.

Це не остання версія сайту, оскільки в майбутнього планується додати

додаткові функції для покращення, зовнішнього вигляду сайту, а також додавання кращого функціоналу сайту для покращення швидкості і зручності в використанні сайту і безпеки для користувачів.

ВИСНОВКИ

У результаті виконання дипломної роботи вдалося створити сайт інтернет-магазину. Для написання бекенду сайту було використано Node.js, що дозволяє розробникам створювати ефективні та масштабовані веб-додатки. Також використовувалися бібліотеки Express та Mongoose, які дозволяють спростити розробку серверної частини та підключення до бази даних MongoDB.

Для розробки фронтенду сайту було використано React, що дозволяє розробникам створювати багатофункціональні та інтерактивні користувацькі інтерфейси. Був використаний вбудований механізм React Context, що дозволив ефективно передавати дані між компонентами, також застосовано React Router та Axios, щоб допомогти в управлінні станом додатку, маршрутизації та забезпеченні зв'язку між сервером та клієнтом.

Надійність та безпека сайту була підвищена за допомогою застосування автентифікації, яка дозволяє контролювати доступ до ресурсів тільки авторизованим користувачам. Для захисту паролів було використано бібліотеку bcrypt, що дозволяє шифрувати паролі та зберігати їх у безпечному форматі.

Окрім цього, у межах реалізованого проєкту вдалося повністю реалізувати ключові функції сучасного інтернет-магазину: систему реєстрації та авторизації користувачів, кошик із динамічним оновленням, обробку та збереження замовлень, адміністративну панель для керування товарами, а також пошукову систему з фільтрацією та сортуванням. Усі дані надійно зберігаються у базі даних MongoDB, а механізми авторизації через JWT-токени та захист паролів за допомогою bcrypt гарантують безпечний доступ до особистої та адміністративної інформації. Таким чином, у процесі виконання дипломної роботи було досягнуто поставленої мети та реалізовано функціональну, захищену й масштабовану платформу для онлайн-продажів.

Узагальнюючи, створення сайту з використанням Node.js та React забезпечує зручний та ефективний процес розробки, а також можливість реалізації різноманітних функціональних можливостей. Однак, необхідно

дотримуватися кращих практик забезпечення безпеки та оптимізації для забезпечення швидкої та безпечної роботи веб-додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мозгова Г.В., Бойко Ю.А. Сайт як інструмент інтернет-маркетингу. *Економіка і суспільство*. 2017. Вип. 9. С. 523-527.

2. Дослідження трендів, видів сайтів та інструментів розробки веб-дизайну. URL:

https://openarchive.nure.ua/server/api/core/bitstreams/ffd32594-1ecc-4aa3-a3cb-13147ec0b394/content (дата звернення: 15.04.2025).

3. Підруцький Д.А., Січко Т.В., Лозинська Л.Ф. Розробка e-commerce сайту для продажу військового спорядження. Актуальні проблеми гуманітарних, технічних і природничих наук: тези доп. учасників XI Міжнар. наук.-практ. конф. студентів, аспірантів і молодих учених, м. Вінниця, 30 квіт. 2025 р. / ДонНУ імені Василя Стуса. Вінниця, 2025. С.34–36.

4. Іваненко Л.М. Маркетплейси як об’єктивний наслідок розвитку електронної комерції. Економіка і організація управління. - 2021.- №4(44). – с. 178-187.

5. Маркотт Е., Responsive Web Design. A Book Apart, 2011. 143 с.

6. Дражниця С. А., Забурмеха Є. М. Електронна комерція: світові тренди та прогноз розвитку в Україні. Вісник Хмельницького національного університету. Серія: Економічні науки. - 2018. - № 6. – с. 69-73

7. І.О. Завадський. Основи баз даних: навч. посіб. Київ, 2011. 192 с.

8. Фрімен Е., Робсон Е., Head First. програмування на JavaScript пер. з анг. Г. Якубовська. Фабула, 2022, 672 с.

9. Самонюк Т. В, Киселев Г. Д. Методи розробки інтерфейсу користувача у веб застосунках. Вісник Університету «Україна». 2020. № 1 (24). С. 210 – 223.

10. Domingues A. L. S., Bianchini S., Costa M. L. S., Ferrari F., Maldonado J. C. Web Application Development Methods: A Comparison. – 2007. URL: https://www.researchgate.net/publication/228937474_Web_application_development_methods_A_comparison (дата звернення: 19.04.2025).

11. Васюта В.В. Адаптивна розробка програмного забезпечення (ASD) / В.В. Васюта, С.Г. Барсуков // Scientific progress: innovations, achievements and prospects : Proceedings of the 4th International scientific and practical conference. – Munich : MDPC Publishing, 2023. – P. 163-164.

12. Дослідження методів виявлення помилок проектування сайтів. URL: https://openarchive.nure.ua/server/api/core/bitstreams/4d99821b-1fb9-442f-9012-7f73225726f8/content (дата звернення: 23.04.2025)

13. Мартін Р. Чиста архітектура: мистецтво розробки програмного забезпечення. - 2019. – 416 с.

14. Граф М. С., Кузьменко О. В. Архітектура, проєктування та безпека веб-орієнтованих інформаційних та комп'ютерних систем. – с.179

15. Андронік О.Л., Воронін А.В. Можливості та загрози електронної комерції в Україні. *Економіка і організація управління*. 2021, Вип. 4(44) С. 118-130.

16. Удосконалення засобів захисту веб-застосунків від несанкціонованого впливу. URL: https://ela.kpi.ua/server/api/core/bitstreams/e72ae542-914e-44e7-8eda-7c37ee1f7c26/content (дата звернення: 26.04.2025)

17. К.М. Краус, Н.М. Краус, О.В. Манжура Електронна комерція та інтернет-торгівля: навчально-методичний посібник. Київ: Аграр Медіа Груп, 2021. 456 с.

18. Нескородєва Т. В., Федоров Є. Є., Січко Т. В., Нескородєва А. Р. Експертні та рекомендаційні системи: навч. посіб. для здобувачів вищої освіти спеціальностей 122 «Комп’ютерні науки», 125 «Кібербезпека», 113 «Прикладна математика». Вінниця: ДонНУ імені Василя Стуса, 2022. 208 с.

19. Madeyski L., Stochmialek M. Architectural Design of Modern Web Applications. – 2005. URL: https://www.researchgate.net/publication/221679095_Architectural_Design_of_Modern_Web_Applications (дата звернення: 28.04.2025).

20. С. О. Лебеденко Навчально-методичний комплекс дисципліни «Електронна комерція». Київ: КПІ ім. Ігоря Сікорського, 2021. 73 с.

21. Ніколаєв І. В., Загреба М. М., Вишневська В. А. Інформаційні послуги електронних торговельних майданчиків у маркетинговій діяльності. Центральноукраїнський науковий вісник. Серія: Економічні науки. 2022. №8(41) С. 56-68.

22. Isoraite M., Miniotiene N. Electronic Commerce: Theory and Practice. – 2018. URL: https://www.researchgate.net/publication/329704574_Electronic_Commerce_Theory_and_Practice (дата звернення: 30.04.2025).

23. Кирнасюк Є. С. Майданюк В. П. Розробка клієнтської частини адаптивної тестувальної системи з фотоконтролем з використанням технологій javascript/typescript та фреймворку ANGULAR. Інноваційні дослідження та перспективи розвитку науки і техніки у XXI столітті : зб. тез доп. учасників Міжнар. наук.-практ. конф. до 30-річчя Приват. вищ. навч. закл. «Міжнар. економ.-гуманітар. ун-т ім. акад. Степана Дем’янчука», м. Рівне, 19 жовт. 2023 р. / ВПНЗ "МЕГУ". Рівне, 2025. Ч 3. С. 182–184.

24. Сучасні фреймворки та бібліотеки для розробки вебзастосунків. URL: https://cutt.ly/mrxjtwrN (дата звернення: 03.05.2025)

25. Гриценко В. Г., Подолян О. М. Використання системи управління версіями Git для організації командної роботи над IT проектом. Інформаційні технології і засоби навчання. 2014. №1 (39). С. 250-263.

26. Іванінська І.І., Абдурайімов Л.Н. Застосування веб-сервісу GITHUB при розробці програмних проектів студентами в процесі навчання. FOSS Lviv-2013 : матеріали II Міжнар. наук.-практ. конф., м. Львів, 18 – 21 квіт. 2013 р. / Львівський національний університет імені Івана Франка. Львів, 2025. С. 66–67.

27. Understanding Web Application Architectures: A Comprehensive Overview of Infrastructure Models and Components. URL: http://medium.com/@gwenilorac/layout-of-web-applications-795b3e8e4c1b (дата звернення: 07.05.2025)

28. Малохвій Є., Бугай В., Молчанов Г., Черних О. Аналітичний огляд та порівняння сучасних javascript рішень для розробки веб-додатків. Системи управління, навігації та зв'язку. 2021. Випуск 4(66). С. 55–58.

29. БЕЗВЕРХИЙ, О., КУЦЕНКО, О. Ефективність застосування бібліотеки react. Інформаційні технології та суспільство. 2023. Випуск 2(4). С. 13–19.

30. Кошова О.П., Ольховська О.В., Тацій Д.С., Олексійчук Ю.Ф., Черненко О.О. Розробка веб-додатків та сервісів на платформі Node.js. Таврійський науковий вісник. Серія: Технічні науки. 2023. Випуск 2. С. 78–89.

31. Розробка та імплементація платформи онлайн-навчання веброзробці для початківців на основі технологій HTML, CSS, та JavaScript з використанням фреймворку Express.js. URL:

https://sci.ldubgd.edu.ua/bitstream/123456789/14396/1/Dyplom_Royuk.pdf (дата звернення: 11.05.2025)

32. Холод І.П. Розробка методики спрощення інтеграції з мікросервісами та сторонніми api у node.js додатках. Проблеми експлуатації та захисту інформаційно-комунікаційних систем : тези доп. XI Міжнар. наук.-практ. конф., м. Київ, 7-9 черв. 2023. / Державний університет інформаційно-комунікаційних технологій. Київ, 2023. С.117–122.

33. Створення Web-застосунку інструментами React у середовищі розробки Visual Studio Code. URL:

http://biblio.umsf.dp.ua/jspui/handle/123456789/6562 (дата звернення: 13.05.2025)

34. Степанов О. В., Клим Г. І. Методологія впровадження інформаційних систем із використанням мікроінтерфейсів для підвищення якості та швидкості їх розробки. Комп’ютерні системи і мережі. 2024. Випуск 6(2). С. 222–231.

35. Шаров С.В., Петровський В.В. Огляд нереляційних баз даних. Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення : тези доп. всеукр. наук. інтернет-конф., м. Тернопіль, 26-27 жовт. / Тернопільський національний економічний університет. М. Тернопіль. 2015. С. 16–18.

36. Дослідження та реалізація адаптивного дизайну веб-сайту для оптимального відображення на різних пристроях з використанням бібліотеки React.js та її екосистеми. URL:

http://elartu.tntu.edu.ua/handle/lib/45509 (дата звернення: 15.05.2025)

37. Палєга Р.В., Карпенко Н.В. Особливості авторизації за допомогою jwt токенів. Інформаційні технології та автоматизація – 2024 : тези доп. XVII Міжнар. наук.-практ. конф., м. Одеса. 30 жовт. – 1 лист. 2024 р. / Одеського національного технологічного університету. Одеса. 2024. С. 208–211.

38. Розробка багатокористувацької карти для гри «Підземелля і дракони» з використанням React і Nodejs. URL:

https://elartu.tntu.edu.ua/handle/lib/45466 (дата звернення: 16.05.2025)

39. Оптимізація рендерингу React компонентів. URL:

https://dspace.znu.edu.ua/xmlui/handle/12345/18443 (дата звернення: 17.05.2025)

40. Розробка веб-додатку SPA з використанням фреймворку React. URL:

https://openarchive.nure.ua/entities/publication/91e0cfc4-f711-441e-a638-bbdcfff965e3 (дата звернення: 18.05.2025)

41. Підруцький Д.А., Січко Т.В. Розробка вебсайту для продажу військового спорядження. Прикладні інформаційні технології: тези доп. учасників VI Всеукр. наук.-практ. конф. здобувачів вищої освіти та молодих вчених, м. Вінниця, 22 трав. 2025 р. / ДонНУ імені Василя Стуса. Вінниця, 2025. С.101–103.

42. Підруцький Д.А., Хмелівський Ю.С. Розробка вебсайту для продажу військового спорядження. Прикладні інформаційні технології: тези доп. учасників V Всеукр. наук.-практ. конф. здобувачів вищої освіти та молодих вчених, м. Вінниця, 24 трав. 2024 р. / ДонНУ імені Василя Стуса. Вінниця, 2025. С.128–130.

ДОДАТКИ

Додаток А

Огляд сайту

Почнемо з головної сторінки, яка відразу ж відкривається, як тільки ми запускаємо реакт додаток, попередньо запустивши сервер.

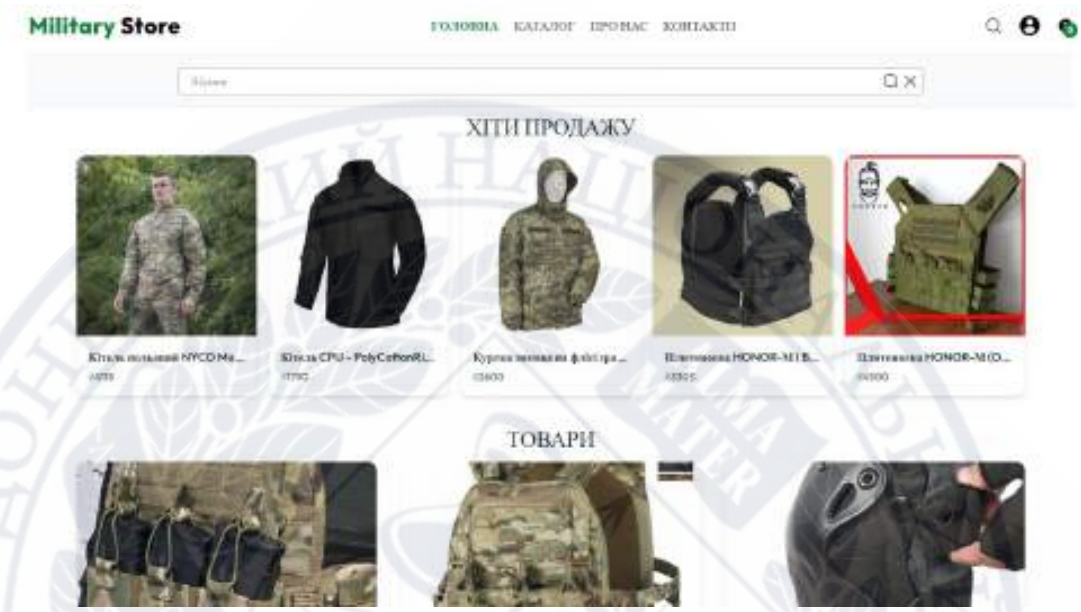

Рисунок 1 – Головна сторінка сайту

Однією з важливих функцій будь якого сайту є простий пошук в текстовому полі, сортування і фільтрація за категоріями

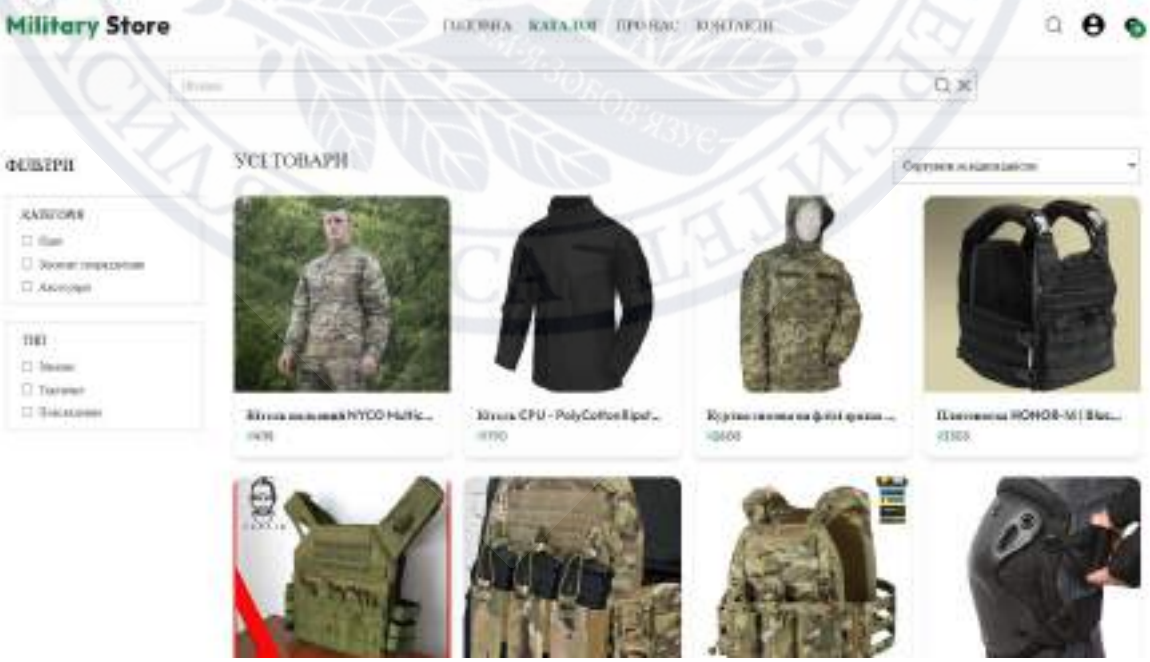

Рисунок 2 – Меню каталогу

Користувач будучи не авторизованим, може побачити детальнішу інформацію про річ яка йому сподобалась. Так при натисканні на фото товару в нього буде показана детальна інформація про товар.

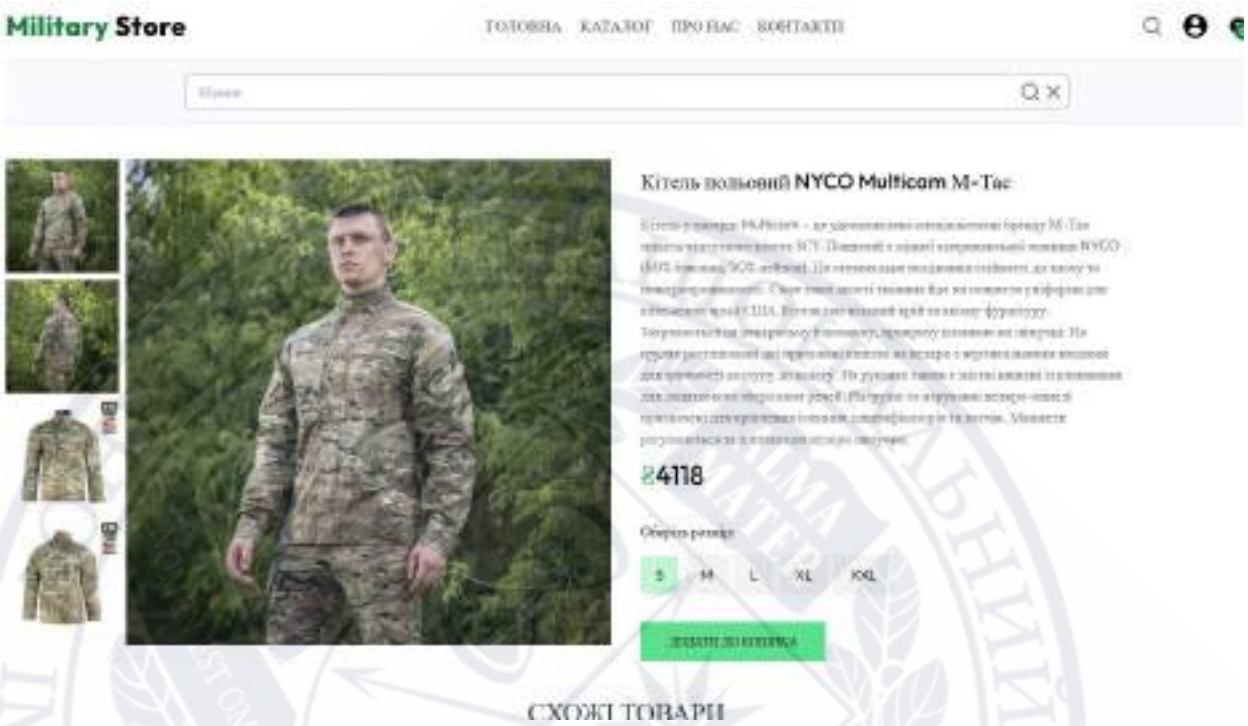

Рисунок 3 – Детальна інформація про товар

Крім того, під інформацією з обраним товаром, знаходяться схожі запропоновані товари.

Рисунок 4 – Схожі товари

Але для того щоб додати обрану річ до кошику потрібно в будь якому випадку ввійти в систему, оскільки в інакшому випадку, при спробі щось додати до кошика, все одно потрібно авторизуватись задля безпеки.

Рисунок 5 – Вікно з авторизуванням на сайті

При реєстрації, буде таке вікно:

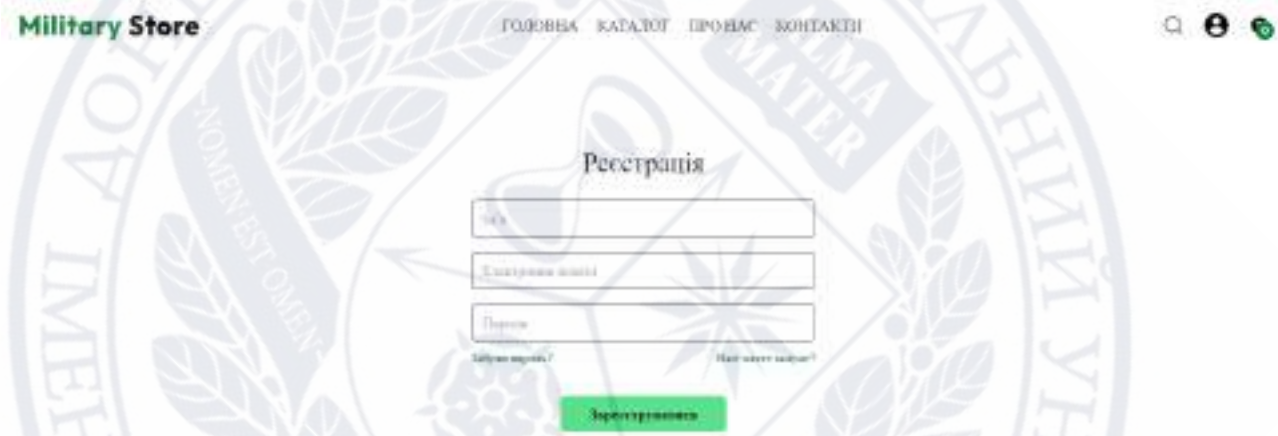

Рисунок 6 – Вікно з реєстрацією

Як було сказано вище, нам також буде доступна функція додавання до кошика

Рисунок 7 – Вікно кошика

Також ми маємо інформацію про загальну кількість речей і їх суму, де при натисканні, будемо мати таке вікно

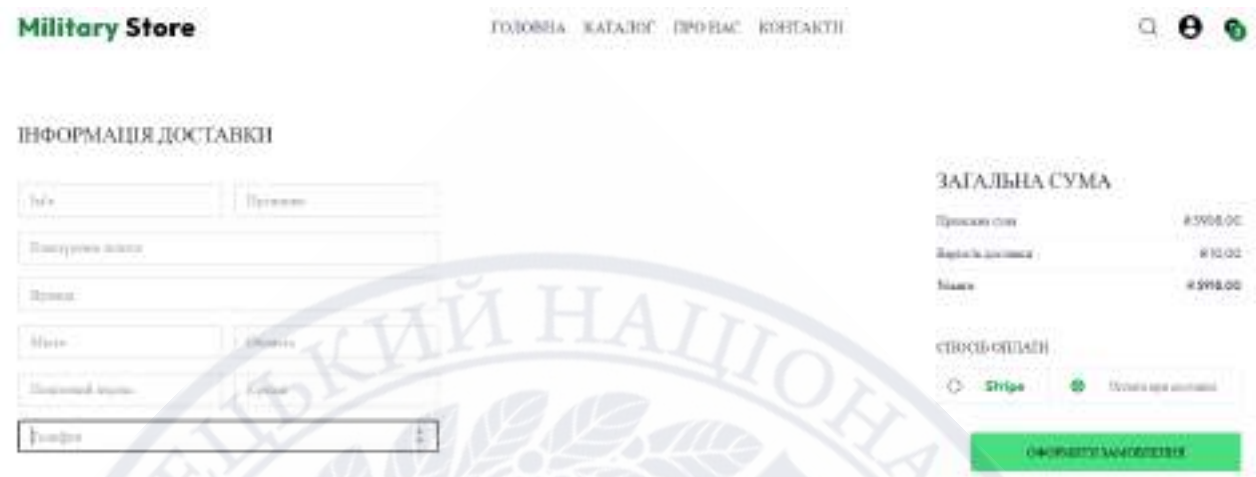

Рисунок 8 – Вікно з заповненням форми доставки

Вікно оплати:

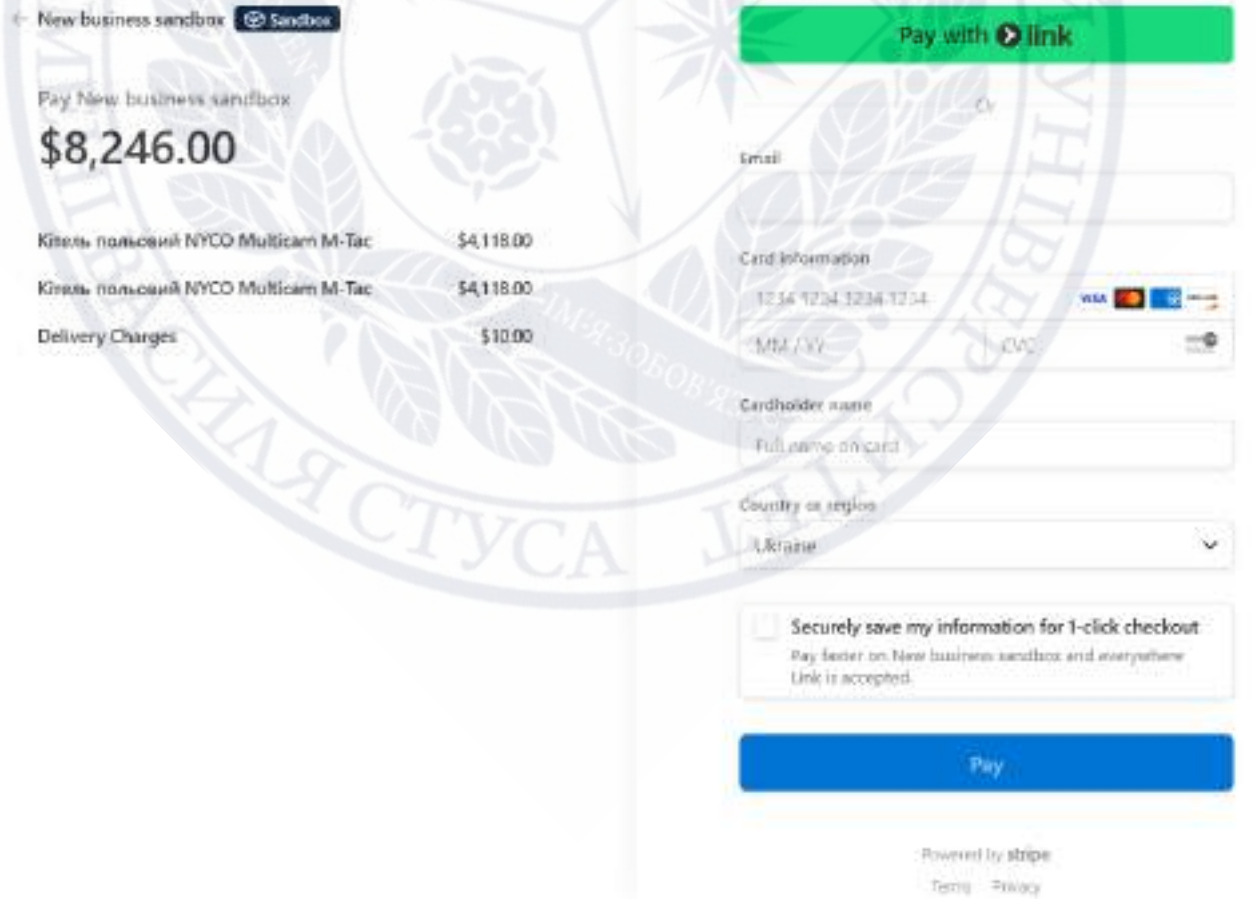

Рисунок 9 – Вікно з оплатою

Далі користувач побачить свої замовлені товари в історії замовлень.

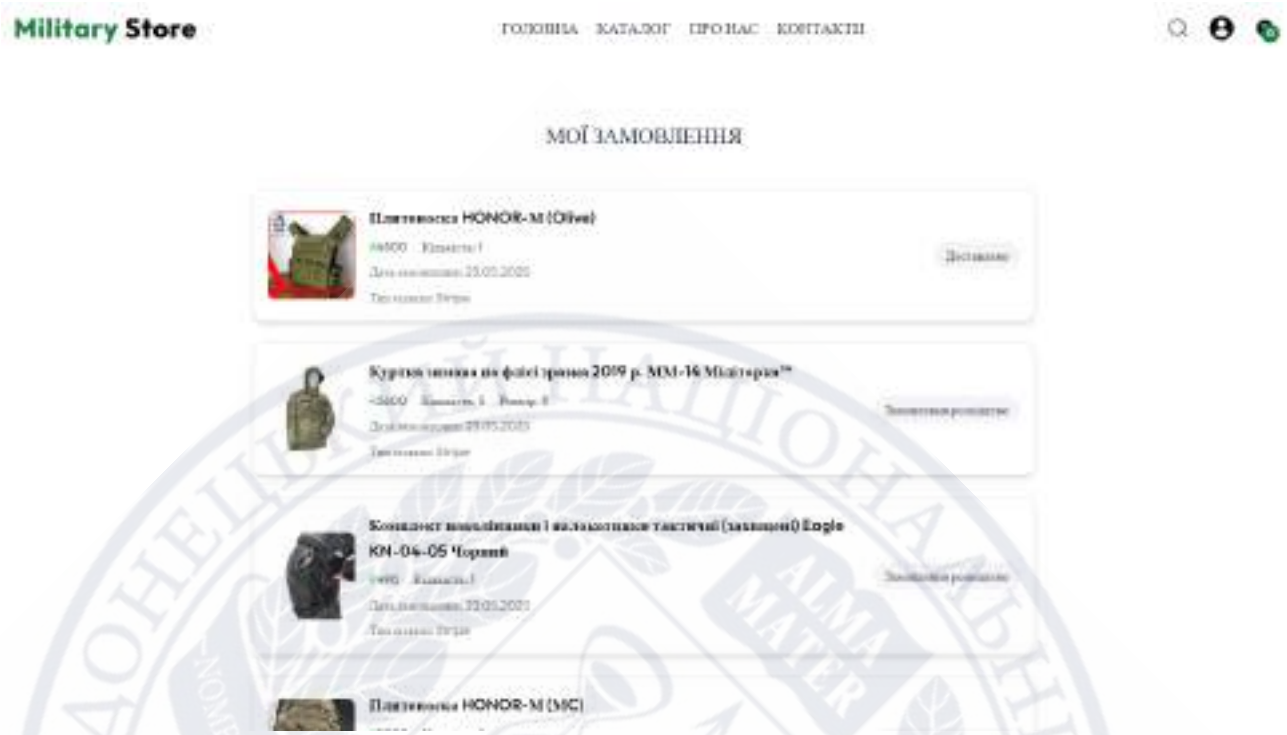

Рисунок 10 – Вікно з інформацією про замовлення

На цьому це всі можливості простого користувача, але є ще можливості входу для адміністратора, який має більше можливостей

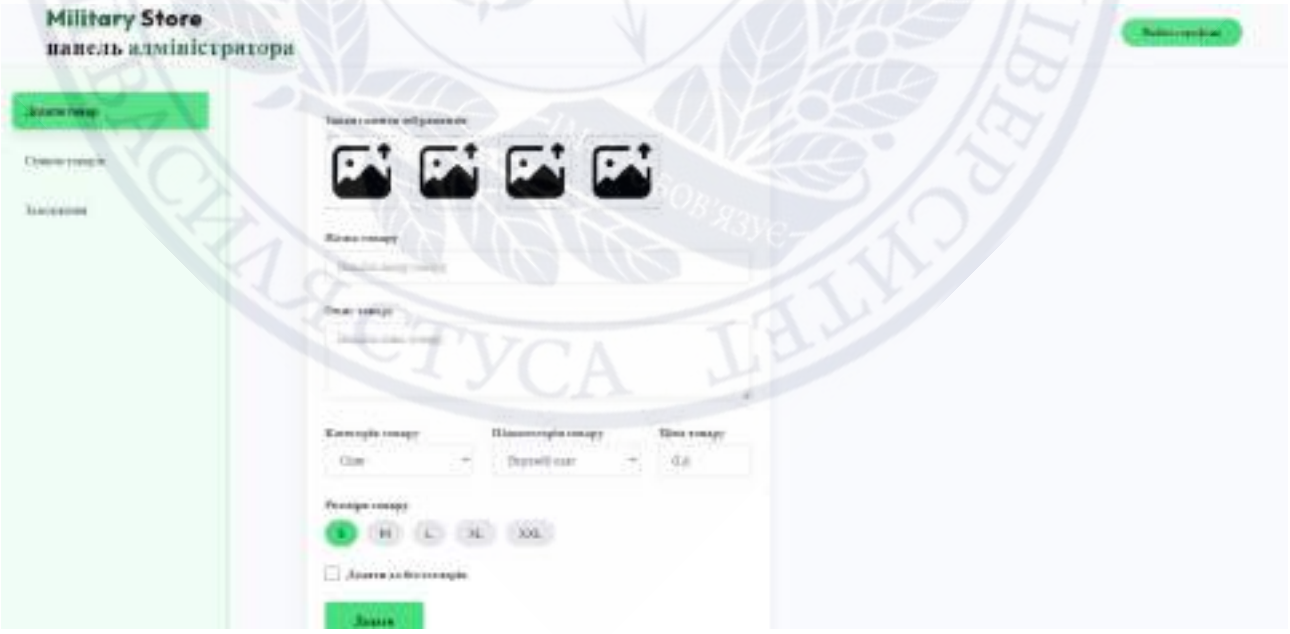

Рисунок 11 – Панель адміністратора

Відповідно перейшовши по панелі управління, адміністратор побачить

Також адміністратор, може перейти до товарів і побачити, додати та змінити інформацію про товари.

Додаток Б

Код програми

Код програми складається з бекенд частини і фронтенд частини, почнемо з бекенд частини. Не беручи о уваги файли package-lock.js і package.js, і папку node_moduls, які утворені завдяки встановленню відповідних бібліотек для роботи серверу, бекенд сайту, ще складається з папки models, routes, controllers, middleware , файлів server.js.

В папці models, знаходяться файли моделей користувачів, замовлень і продуктів, приведу приклад моделі товарів:

```
import mongoose, { Mongoose } from "mongoose";
const productSchema = new mongoose.Schema({
    name: {type: String, required: true},
    description: {type: String, required: true},
    price: {type: Number, required: true},
    image: {type: Array, required: true},
    category: {type: String, required: true},
    subCategory: {type: String, required: true},
    bestseller: { type: Boolean, default: false },
    sizes: {type: Array},
    date: {type: Number, required: true},
})
const productModel = mongoose.models.product || mongoose.model('product',
productSchema);
export default productModel;
```

В папці Routes знаходяться файли маршрутизатори для обробки запитів, наприклад маршрутизатор запитів користувача, має такий вигляд:

```
import express from 'express'
import { loginUser, registerUser, adminLogin } from
'../controllers/userController.js'

const userRouter = express.Router()

userRouter.post('/register', registerUser)
userRouter.post('/login', loginUser)
userRouter.post('/admin', adminLogin)

export default userRouter;
```

Код реалізує три основні функції автентифікації у Node.js контролері користувача:

```
import userModel from "../models/userModel.js"
import validator from "validator"
import bcrypt from "bcryptjs" //
import jwt from "jsonwebtoken"

const createToken = (id) => {
    return jwt.sign({id}, process.env.JWT_SECRET)
}
const loginUser = async (req, res) => {
    try{
        const {email, password} = req.body;
        const user = await userModel.findOne({email})
        if(!user){
            return res.json({success: false, message:"Такого користувача не існує
"});
        }
        const isMatch = await bcrypt.compare(password, user.password)
        if(isMatch){
            const token = createToken(user._id)
            res.json({success: true, token})
        }
        else{
            res.json({success: false, msg: "Неправильні облікові дані"})
        }
        }
    catch (error){
        console.log(error);
        res.json({success: false, message:error.message})
    }
}
const registerUser = async (req, res) => {
    try{
        const {name, email, password} = req.body;
        const exists = await userModel.findOne({email})
        if(exists){
            return res.json({success: false, message:"Такий користувач вже
існує"})}
        if(!validator.isEmail(email)){
            return res.json({success: false, message:"Будь ласка
введіть  електронну пошту"})}
        if(password.length < 8){
            return res.json({success: false, message:"Будь ласка введіть складніший
пароль"})}
        const salt = await bcrypt.genSalt(10)
        const hashedPassword = await bcrypt.hash(password, salt)
        const newUser = new userModel({
            name,
            email,
            password: hashedPassword})
        const user = await newUser.save()
```

```
        const token = createToken(user._id)
        res.json({success: true, token})}
    catch(error){
        console.log(error);
        res.json({success: false, message:error.message})}}
const adminLogin = async (req,res) => {
    try{
        const {email, password} = req.body

        if(email === process.env.ADMIN_EMAIL && password ===
process.env.ADMIN_PASSWORD){
            const token = jwt.sign(email+password, process.env.JWT_SECRET)
            res.json({success: true, token})}
        else{
            res.json({success: false, message:"Невірні облікові дані"})}}
    catch (error){
        console.log(error);
        res.json({success: false, message:error.message})}}

export {loginUser, registerUser, adminLogin}
```

Файл серверу:

```
import express from 'express'
import cors from 'cors'
import 'dotenv/config'
import connectDB from './config/mongodb.js'
import connectCloudinary from './config/cloudinary.js'
import userRouter from './routes/userRoute.js'
import productRouter from './routes/productRoute.js'
import cartRouter from './routes/cartRoute.js'
import orderRouter from './routes/orderRoute.js'

const app = express()
const port = process.env.PORT || 4000
connectDB()
connectCloudinary()
app.use(express.json())
app.use(cors())
app.use('/api/user', userRouter)
app.use('/api/product', productRouter)
app.use('/api/cart',cartRouter)
app.use('/api/order', orderRouter)

app.get('/', (req, res)=>{
    res.send("API Working")
})
app.listen(port, ()=>{
    console.log('Server started on PORT : ' + port)
})
```

Цей код запускає сервер на Express для інтернет-магазину. Він:

- Підключає базу даних MongoDB і сервіс Cloudinary для перетворення зображень в фото.
- Додає middleware для JSON-запитів і дозволяє CORS.
- Реєструє маршрути: /api/user, /api/product, /api/cart, /api/order.
- Виводить "API Working" на головній сторінці /.
- Слухає порт (з .env або 4000) і запускає сервер.

Фронтенд частина, крім основних файлів, які були створені під час створення реакт додатку, і встановлення відповідних бібліотек, має в собі основний файл з наповненням App.jsx, а також папки Context, Pages і Components.

Файл App.jsx:

```jsx
import React from 'react'
import {Routes, Route} from 'react-router-dom'
import Catalog from './pages/Catalog'
import About from './pages/About'
import Contact from './pages/Contact'
import Product from './pages/Product'
import Cart from './pages/Cart'
import Login from './pages/Login'
import PlaceOrder from './pages/PlaceOrder'
import Orders from './pages/Orders'
import Navbar from './components/Navbar'
import Home from './pages/Home'
import SearchBar from './components/SearchBar'
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/ReactToastify.css'
import Verify from './pages/Verify'

const App = () => {
  return (
    <div className='px-4 sm:px-[5vw] md:px-[7vw] lg:px-[9vw]'>
      <ToastContainer/>
      <Navbar/>
      <SearchBar/>
      <Routes>
      <Route path='/' element={<Home/>}/>
      <Route path='/catalog' element={<Catalog/>} />
      <Route path='/about' element={<About/>} />
      <Route path='/contact' element={<Contact/>} />
```

```
      <Route path='/product/:productId' element={<Product/>} />
      <Route path='/cart' element={<Cart/>}/>
      <Route path='/login' element={<Login/>}/>
      <Route path='/place-order' element={<PlaceOrder/>}/>
      <Route path='/orders' element={<Orders/>} />
      <Route path='/verify' element={<Verify/>} />
      </Routes>
    </div>
  )
}

export default App
```

В папці pages, перебувають файли, які відповідають за основний контент сторінок, наприклад вміст файлу Product.jsx:

```
import React, { useContext, useEffect, useState } from "react";
import { useParams } from "react-router-dom";
import { ShopContext } from "../context/ShopContext";
import RelatedProduct from "../components/RelatedProduct";

const Product = () => {
  const { productId } = useParams();
  const { products, currency, addToCart } = useContext(ShopContext);
  const [productData, setProductData] = useState(null);
  const [image, setImage] = useState("");
  const [size, setSize] = useState("");

  useEffect(() => {
    const product = products.find((item) => item._id === productId);
    if (product) {
      setProductData(product);
      setImage(product.image[0]);
    }
  }, [productId, products]);

  return productData ? (
    <div className="pt-10 transition-opacity ease-in duration-500 opacity-100">
      <div className="flex flex-col sm:flex-row gap-8">
        <div className="flex-1 flex flex-col-reverse sm:flex-row gap-2">
          <div className="flex sm:flex-col overflow-x-auto sm:overflow-y-scroll
justify-start sm:w-[18.7%] w-full">
            {productData.image.map((item, index) => (
              <img
                key={index}
                src={item}
                alt=""
                onClick={() => setImage(item)}
                className="w-[24%] sm:w-full sm:mb-2 flex-shrink-0 cursor-pointer"
```

```
            />
          ))}
        </div>
        <div className="w-full sm:w-[80%]">
          <img className="w-full h-auto" src={image} alt="Основне зображення
товару" />
        </div>
      </div>

      <div className="flex-1">
        <h1 className="font-medium text-2xl mt-2">{productData.name}</h1>
        <p className="mt-5 text-gray-500 md:w-4/5">{productData.description}</p>
        <p className="mt-3 text-3xl font-medium">
          <span className="text-green-600">{currency}</span>{productData.price}
        </p>

        {productData.sizes.length > 0 ? (
          <div className="flex flex-col gap-4 my-8">
            <p>Оберіть розмір:</p>
            <div className="flex gap-2">
              {productData.sizes.map((item, index) => (
                <button
                  key={index}
                  onClick={() => setSize(item)}
                  className={`border py-2 px-4 bg-gray-100 ${item === size ? 'bg-
green-200 text-black' : ''}`}
                >
                  {item}
                </button>
              ))}
            </div>
          </div>
        ) : (
          <div className="my-8 text-black">Цей товар не має вибору розмірів</div>
        )}

        <button
          onClick={() => addToCart(productData._id, size, productData.sizes)}
          className="bg-green-400 text-black px-8 py-3 text-sm active:bg-white"
        >
          ДОДАТИ ДО КОШИКА
        </button>
      </div>
    </div>

    <RelatedProduct
      category={productData.category}
      subcategory={productData.subcategory}
      currentProductId={productData._id}
    />
```

```
    </div>
  ) : (
    <div className="opacity-0"></div>
  );
};

export default Product;
```

Файл контексту загалом тримає всі змінні та їхні стани:

```
import { createContext, useEffect, useState } from "react";
import { toast } from "react-toastify";
import {useNavigate} from 'react-router-dom'
import axios from 'axios'
export const ShopContext = createContext();

const ShopContextProvider = (props) => {

  const currency = "₴";
  const delivery_fee = 10;
  const backendUrl = import.meta.env.VITE_BACKEND_URL
  const [search, setSearch] = useState("");
  const [showSearch, setShowSearch] = useState(false);
  const [cartItems, setCartItems] = useState({});
  const navigate = useNavigate()
  const [token, setToken] = useState('')
  const [products, setProducts] = useState([])

  const addToCart = async (itemId, size, sizes = []) => {
    if (sizes.length > 0 && !size) {
      toast.error("Оберіть розмір товару");
      return;
    }
    let cartData = structuredClone(cartItems);
    if (cartData[itemId]) {
      if (cartData[itemId][size]) {
        cartData[itemId][size] += 1;
      } else {
        cartData[itemId][size] = 1}
    } else {
      cartData[itemId] = {};
      cartData[itemId][size] = 1}
    setCartItems(cartData);
    if (token) {
      try {
        await axios.post(backendUrl + '/api/cart/add', { itemId, size }, { headers:
{ token } });
      } catch (error) {
        console.log(error);
        toast.error(error.message)}}}
```

```
  const getCartCount = () => {
    let totalCount = 0;
    for (const items in cartItems) {
      for (const item in cartItems[items]) {
        try {
          if (cartItems[items][item] > 0) {
            totalCount += cartItems[items][item]}
        } catch (error) {}}}
    return totalCount}
  const updateQuantity = async (itemId, size, quantity) => {
    let cartData = structuredClone(cartItems);
    cartData[itemId][size] = quantity;
    setCartItems(cartData);
    if (token) {
      try {
        await axios.post(backendUrl + '/api/cart/update', {itemId, size, quantity},
{headers:{token}})
      } catch (error) {
        console.log(error);
        toast.error(error.message)}}};

  const getUserCart = async (token)=>{
    try {
      const response = await axios.post(backendUrl + '/api/cart/get', {}, {headers:
{token}})
      if (response.data.success) {
        setCartItems(response.data.cartData)}
    } catch (error) {
      console.log(error);
      toast.error(error.message)}}
  const getCartAmount = () => {
    let totalAmount = 0;
    for (const items in cartItems) {
      let itemInfo = products.find((product) => product._id === items);
      for (const item in cartItems[items]) {
        try {
          if (cartItems[items][item] > 0) {
            totalAmount += itemInfo.price * cartItems[items][item]}
        } catch (error) {}}}
    return totalAmount}

  const getProductsData = async ()=>{
    try {
      const response = await axios.get(backendUrl+'/api/product/list')
      if(response.data.success){
        setProducts(response.data.products)
      } else{
        toast.error(response.data.message)
      }
    } catch (error) {
```

```
      console.log(error);
      toast.error(error.message)
    }
  }

  useEffect(()=>{
    getProductsData()
  }, [])

  useEffect(()=>{
    if (!token && localStorage.getItem('token')) {
      setToken(localStorage.getItem('token'))
      getUserCart(localStorage.getItem('token'))
    }
  })
  const value = {
    products,
    currency,
    delivery_fee,
    search,
    setSearch,
    showSearch,
    setShowSearch,
    cartItems,
    addToCart,
    getCartCount,
    updateQuantity,
    getCartAmount,
    navigate, backendUrl,
    setToken, token,
    setCartItems
  };
  return (
    <ShopContext.Provider value={value}>{props.children}</ShopContext.Provider>
  );
};

export default ShopContextProvider;
```

В папці Components знаходяться файли які часто застосовуються на сторінках сайту, наприклад файл RelatedProducts.jsx, який пропонує товари на основі категорій та підкатегорій обраного товару:

```
import React, { useContext, useEffect, useState } from "react";
import { ShopContext } from "../context/ShopContext";
import Title from "../components/Title";
import ProductItem from "../components/productItem";

const RelatedProduct = ({ category, subcategory, currentProductId }) => {
```

```
  const { products } = useContext(ShopContext);
  const [relatedProducts, setRelatedProducts] = useState([]);

  useEffect(() => {
    if (products.length > 0) {
      const filtered = products
        .filter(
          (item) =>
            item.category === category &&
            item.subcategory === subcategory &&
            item._id !== currentProductId
        )
        .slice(0, 5);
      setRelatedProducts(filtered);
    }
  }, [products, category, subcategory, currentProductId]);

  return (
    <div className="my-8">
      <div className="text-center text-3xl py-2">
        <Title text1="СХОЖІ" text2="ТОВАРИ" />
      </div>
      <div className="grid grid-cols-2 sm:grid-cols-3 md:grid-cols-4 lg:grid-cols-5
gap-4 gap-y-6">
        {relatedProducts.map((item) => (
          <ProductItem
            key={item._id}
            id={item._id}
            name={item.name}
            price={item.price}
            image={item.image}
          />
        ))}
      </div>
    </div>
  );
};

export default RelatedProduct;
```

Щодо фротенду панелі адміністратора, вона має свої папки components та pages і свій App.jsx файл.

Файл App.jsx:

```
import React, { useEffect, useState } from "react";
import Navbar from "./components/Navbar";
import SideBar from "./components/SideBar";
import { Route, Routes } from "react-router-dom";
```

```jsx
import Add from "./pages/Add";
import List from "./pages/List";
import Orders from "./pages/Orders";
import Login from "./components/Login";
import { ToastContainer } from 'react-toastify';
import 'react-toastify/ReactToastify.css'

export const backendUrl = import.meta.env.VITE_BACKEND_URL
export const currency = '₴'
const App = () => {
  const [token, setToken] =
useState(localStorage.getItem('token')?localStorage.getItem('token'):'');
  useEffect(()=>{
    localStorage.setItem('token', token)
  }, [token])
  return (
    <div className=" bg-gray-50 min-h-screen">
      <ToastContainer/>
      {token === "" ? <Login setToken={setToken} /> : <>
          <Navbar setToken={setToken} />
          <hr />
          <div className=" flex w-full">
            <SideBar />
            <div className="w-[70%] mx-auto ml-[max(5vw, 25px)] my-8 text-gray-600
text-base">
              <Routes>
                <Route path="/add" element={<Add token={token} />} />
                <Route path="/list" element={<List  token={token}/>} />
                <Route path="/orders" element={<Orders token={token} />} />
              </Routes>
            </div>
          </div>
        </>
      }
    </div>)};
export default App;
```

прикладом файлів папки pages є файл Add.jsx:

```jsx
import React, { useState } from 'react';
import { assets } from '../assets/assets';
import axios from 'axios';
import { backendUrl } from '../App';
import { toast } from 'react-toastify';

const Add = ({ token }) => {
  const [image1, setImage1] = useState(false);
  const [image2, setImage2] = useState(false);
  const [image3, setImage3] = useState(false);
  const [image4, setImage4] = useState(false);
```

```
  const [name, setName] = useState('');
  const [description, setDescription] = useState('');
  const [price, setPrice] = useState('');
  const [category, setCategory] = useState('Одяг');
  const [subCategory, setSubCategory] = useState('Верхній одяг');
  const [bestseller, setBestseller] = useState(false);
  const [sizes, setSizes] = useState([]);

  const onSubmitHandler = async (e) => {
    e.preventDefault();
    try {
      const formData = new FormData();
      formData.append('name', name);
      formData.append('description', description);
      formData.append('price', price);
      formData.append('category', category);
      formData.append('subCategory', subCategory);
      formData.append('bestseller', bestseller ? 'true' : 'false');

      if (sizes.length > 0) {
        formData.append('sizes', JSON.stringify(sizes));
      }

      image1 && formData.append('image1', image1);
      image2 && formData.append('image2', image2);
      image3 && formData.append('image3', image3);
      image4 && formData.append('image4', image4);

      const response = await axios.post(backendUrl + '/api/product/add', formData,
{ headers: { token } });

      if (response.data.success) {
        toast.success(response.data.message);
        setName('');
        setDescription('');
        setImage1(false);
        setImage2(false);
        setImage3(false);
        setImage4(false);
        setPrice('');
        setSizes([]);
      } else {
        toast.error(response.data.message);
      }
    } catch (error) {
      console.log(error);
      toast.error(error.message);
    }
  };
```

```
  return (
    <form onSubmit={onSubmitHandler} className="flex flex-col w-full max-w-[600px] gap-6 p-6 bg-white rounded-lg shadow-lg">
      <div>
        <p className="mb-2 font-semibold text-gray-700">Завантажити зображення</p>
        <div className="flex gap-4">
          {[image1, image2, image3, image4].map((img, idx) => (
            <label
              key={idx}
              htmlFor={`image${idx + 1}`}
              className="w-24 h-24 border-2 border-dashed border-gray-300 rounded-md cursor-pointer flex items-center justify-center overflow-hidden hover:border-green-400 transition"
            >
              <img
                className="object-cover w-full h-full"
                src={!img ? assets.upload_area : URL.createObjectURL(img)}
                alt={`Зображення ${idx + 1}`}
              />
              <input
                onChange={(e) => {
                  const file = e.target.files[0];
                  if (!file) return;
                  if (idx === 0) setImage1(file);
                  if (idx === 1) setImage2(file);
                  if (idx === 2) setImage3(file);
                  if (idx === 3) setImage4(file);
                }}
                type="file"
                id={`image${idx + 1}`}
                hidden
              />
            </label>
          ))}
        </div>
      </div>

      <div>
        <label className="block mb-1 font-semibold text-gray-700" htmlFor="name">
          Назва товару
        </label>
        <input
          id="name"
          onChange={(e) => setName(e.target.value)}
          value={name}
          className="w-full px-4 py-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2 focus:ring-green-400"
          type="text"
          placeholder="Введіть назву товару"
```

```
          required
        />
      </div>

      <div>
        <label className="block mb-1 font-semibold text-gray-700"
htmlFor="description">
          Опис товару
        </label>
        <textarea
          id="description"
          onChange={(e) => setDescription(e.target.value)}
          value={description}
          className="w-full px-4 py-2 border border-gray-300 rounded-md
focus:outline-none focus:ring-2 focus:ring-green-400 resize-y min-h-[100px]"
          placeholder="Введіть опис товару"
          required
        />
      </div>

      <div className="flex flex-col sm:flex-row gap-6">
        <div className="flex-1">
          <label className="block mb-1 font-semibold text-gray-700"
htmlFor="category">
            Категорія товару
          </label>
          <select
            id="category"
            onChange={(e) => setCategory(e.target.value)}
            className="w-full px-4 py-2 border border-gray-300 rounded-md
focus:outline-none focus:ring-2 focus:ring-green-400"
            value={category}
          >
            <option value="Одяг">Одяг</option>
            <option value="Захисне спорядження">Захисне спорядження</option>
            <option value="Аксесуари">Аксесуари</option>
          </select>
        </div>

        <div className="flex-1">
          <label className="block mb-1 font-semibold text-gray-700"
htmlFor="subCategory">
            Підкатегорія товару
          </label>
          <select
            id="subCategory"
            onChange={(e) => setSubCategory(e.target.value)}
            className="w-full px-4 py-2 border border-gray-300 rounded-md
focus:outline-none focus:ring-2 focus:ring-green-400"
            value={subCategory}
```

```
          >
            <option value="Верхній одяг">Верхній одяг</option>
            <option value="Зимове">Зимове</option>
            <option value="Тактичне">Тактичне</option>
            <option value="Повсякденне">Повсякденне</option>
          </select>
        </div>

        <div className="w-full sm:w-[120px]">
          <label className="block mb-1 font-semibold text-gray-700"
htmlFor="price">
            Ціна товару
          </label>
          <input
            id="price"
            onChange={(e) => setPrice(e.target.value)}
            value={price}
            className="w-full px-4 py-2 border border-gray-300 rounded-md
focus:outline-none focus:ring-2 focus:ring-green-400"
            type="number"
            placeholder="100"
            min="0"
            step="0.01"
            required
          />
        </div>
      </div>

      <div>
        <p className="mb-2 font-semibold text-gray-700">Розміри товару</p>
        <div className="flex gap-3 flex-wrap">
          {['S', 'M', 'L', 'XL', 'XXL'].map((size) => (
            <button
              key={size}
              type="button"
              onClick={() =>
                setSizes((prev) =>
                  prev.includes(size) ? prev.filter((item) => item !== size) :
[...prev, size]
                )
              }
              className={`px-4 py-1 rounded-full border transition ${
                sizes.includes(size)
                  ? 'bg-green-400 text-black border-green-400'
                  : 'bg-gray-200 text-gray-700 border-transparent hover:bg-green-
200'
              }`}
            >
              {size}
            </button>
```

```
          ))}
        </div>
      </div>

      <div className="flex items-center gap-2">
        <input
          onChange={() => setBestseller((prev) => !prev)}
          checked={bestseller}
          type="checkbox"
          id="bestseller"
          className="w-5 h-5 accent-green-500 cursor-pointer"
        />
        <label className="cursor-pointer font-semibold text-gray-700"
htmlFor="bestseller">
          Додати до бестселерів
        </label>
      </div>

      <button
        type="submit"
        className="w-full sm:w-32 bg-green-400 hover:bg-green-500 text-black font-
semibold py-3 rounded-md transition"
      >
        Додати
      </button>
    </form>
  );
};

export default Add;
```

Щодо папки components її прикладом є файл Navbar.jsx:

```
const Navbar = ({ setToken }) => {
  return (
    <div className="flex items-center py-2 px-[4%] justify-between">
      <div>
        <p className="text-2xl md:text-3xl font-extrabold text-green-800 tracking-
wide">
          Military <span className="text-black">Store</span>
        </p>
        <p className="text-2xl md:text-3xl font-extrabold text-black tracking-
wide">
          панель <span className="text-2xl md:text-3xl font-extrabold text-green-
800 tracking-wide">адміністратора</span>
        </p>
      </div>
      <button
        onClick={() => setToken("")}
```

```
        className="bg-green-400 text-black px-5 py-2 sm:px-7 rounded-full text-xs
sm:text-sm hover:bg-green-500 transition-colors"
      >
        Вийти з профілю
      </button>
    </div>
  );
};

export default Navbar;
```