НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ
кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

“До захисту допущено”
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА

“___”_____2024 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
“Цифрові технології в енергетиці”
зі спеціальності *122 “Комп’ютерні науки”*

на тему: **Веб-портал підбору військової амуніції**

Виконав:
студент IV курсу, групи ТР-01

БАГНІЙ Антон Іванович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник:
доцент каф. цифрових технологій в енергетиці, к.т.н,
Ірина МИХАЙЛОВА _____
(посада, вчене звання ,науковий ступінь, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент: *доцент каф. теплової та альтернативної енергетики,*
доцент, к.т.н., Ірина ФУРТАТ _____
(посада, вчене звання, науковий ступінь, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль: старший викладач каф. цифрових технологій в енергетиці
Ольга БЕСПАЛА _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА_

«__»_____2024 р.

ЗАВДАННЯ
на дипломну роботу студенту

БАГНІЮ Антону Івановичу
(прізвище, ім’я, по батькові)

1. Тема роботи **Веб-портал підбору військової амуніції**

керівник роботи ***Михайлова Ірина Юріївна, доцент, к.т.н.***
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 202__ р. №_____

2. Термін подання роботи студентом **07.06.2024 року**

3. Вихідні дані до роботи мова програмування TypeScript, фреймворк Next.js, база даних MongoDB, бібліотека Zustand, фреймворк Tailwind CSS, колекція компонентів Shadcn, середовище розробки WebStorm, інструмент для тестування Postman

4. Перелік питань, які потрібно розробити
1) провести аналіз існуючих веб-сервісів підбору військової амуніції;
2) розробити архітектуру та структуру веб-порталу для підбору військової амуніції;

3) розробити веб-портал підбору військової амуніції;
4) комплексно протестувати розроблений веб-портал.

5. Орієнтований перелік ілюстративного матеріалу актуальність роботи, мета та завдання роботи, функціональні можливості, аналіз існуючих веб-порталів підбору військової амуніції, засоби розробки, архітектура веб-порталу, діаграма прецедентів, схема бази даних, основні сторінки веб-порталу

6. Дата видачі завдання «15» вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1.	Затвердження теми роботи	15.09.23	Виконано
2.	Вивчення та аналіз задачі	17-20.04.24	Виконано
3.	Розробка архітектури та загальної структури системи	21-25.04.24	Виконано
4.	Розробка частин окремих підсистем	25.04-02.05.24	Виконано
5.	Програмна реалізація системи	03-07.05.24	Виконано
6.	Оформлення пояснювальної записки	08-12.05.24	Виконано
7.	Захист програмного продукту	13-17.05.24	Виконано
8.	Передзахист	03-06.06.24	Виконано
9.	Подання готової роботи на кафедру	07.06.2024	Виконано
10.	Захист	17-21.06.2024	

Студент _____ **Антон БАГНІЙ**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ **Ірина МИХАЙЛОВА**
(підпис) (ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 61 сторінці, містить 53 ілюстрації, 18 джерел в переліку посилань.

Мета роботи – розробка веб-порталу підбору військової амуніції використовуючи сучасні технології для розробки та найкращі практики. Програмний продукт створений зважаючи на актуальні потреби користувачів та враховуючи переваги і недоліки аналогічних сервісів. Веб-портал має зручні інструменти для його адміністрування власниками сервісу та повний функціонал для підбору військової амуніції у користувачів.

Засоби розробки: мова програмування TypeScript, сучасний веб-фреймворк Next.js, база даних MongoDB, бібліотека Zustand, фреймворк Tailwind CSS, колекція компонентів shadcn, інтегроване середовище розробки WebStorm.

Результат роботи – розроблений веб-портал, який надає можливості аутентифікувати користувачів, додавати, видаляти та змінювати товари, переглядати замовлення, переглядати користувачів та змінювати їхні дані для адміністратора; переглядати товари, фільтрувати за критеріями, виконувати пошук, додавати в кошик та до обраного, оформлювати замовлення для користувачів.

Ключові слова: ВЕБ-ПОРТАЛ, NEXT.JS, BACK-END, FRONT-END, ТОВАРИ, ВІЙСЬКОВА АМУНІЦІЯ, ПІДБІР.

ABSTRACT

The thesis is 61 pages long, contains 53 illustrations, 18 sources in the list of references.

The purpose of the work is to develop a web portal for the selection of military ammunition using modern development technologies and best practices. The software product was created based on the current needs of users and taking into account the advantages and disadvantages of similar services. The web portal has convenient tools for its administration by the service owners and full functionality for users to select military ammunition.

Development tools: TypeScript programming language, modern Next.js web framework, MongoDB database, Zustand library, Tailwind CSS framework, shadcn component collection, WebStorm integrated development environment.

The result of the work is a developed web portal that provides the ability to authenticate users, add, delete and modify products, view orders, view users and change their data for the administrator; view products, filter by criteria, perform searches, add to cart and to favourites, place orders for users.

Keywords: WEB PORTAL, NEXT.JS, BACK-END, FRONT-END, PRODUCTS, MILITARY AMMUNITION, SELECTION.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

Back-end – серверна частина програмного застосунку.

Front-end – клієнтська частина застосунку.

API – прикладний програмний інтерфейс (англ. Application Programming Interface).

SEO – пошукова оптимізація сайту (англ. Search Engine Optimization).

DOM – об’єктна модель документа (англ. Document Object Model).

SSR – рендеринг на стороні серверу (англ. Server-side Rendering).

SSG – статична генерація сайту (англ. Static Site Generation).

JS – мова програмування JavaScript.

TS – мова програмування TypeScript.

CSS – мова стилів (англ. Cascading Style Sheets).

БД – база даних.

UX – досвід користувача (англ. User Experience).

ВСТУП

У сучасному світі інформаційних технологій та ринкових відносин відбувається розвиток електронної комерції. Все більше підприємців починають свою комерційну діяльність в інтернеті, створюючи інтернет-магазини. Вони мають значні переваги над реальними магазинами, а саме менші вкладення для початку бізнесу, не потрібно витрачати кошти на оренду приміщення та кількість клієнтів значно більша, так як в інтернет-магазині товар можуть замовляти люди з будь-якого куточку країни, або навіть світу. З точки зору клієнтів традиційні способи закупівлі можуть мати певні труднощі такі як, обмежений асортимент товару, довгі терміни поставки потрібної продукції, відсутність можливості порівняння цін, або підбір займає багато часу. Більшість цих проблем вирішують інтернет-магазини.

Оскільки в Україні вже довгий час йде війна, військова амуніція користується значним попитом і зростає потреба у високоякісному та своєчасному забезпеченні військових необхідною амуніцією. В таких умовах зручність та швидкість отримання цих речей відіграє важливу роль у забезпеченні боєздатності військових. Забезпечення швидкого та зручного доступу до якісного військового спорядження через інтернет-магазин дозволяє оперативно задовольняти потреби цілих підрозділів та волонтерських організацій.

В Україні вже існує багато інтернет-магазинів військової амуніції, але в них є недоліки. Багато з них мають застарілі інтерфейси, низьку продуктивність, відсутність інтерактивних фільтрів, зручного оформлення замовлення та достатньої інформативності. Найважливішим в таких веб-порталах та інтернет-магазинах є функціональність та зручність користування, а саме інтуїтивно зрозумілий інтерфейс, простота навігації та швидкість доступу, ці аспекти відіграють вирішальне значення для залучення та утримання клієнтів.

Таким чином, актуальним є завдання створення веб-порталу підбору військової амуніції, що дозволить значно підвищити ефективність та зручність процесу закупівлі необхідного спорядження. Такий веб-портал забезпечить користувачів швидким доступом до широкого асортименту товарів, полегшить

порівняння різних пропозицій та дозволить здійснювати покупки з будь-якого місця та у будь-який час. Інтерактивні фільтри та зручний інтерфейс дозволить користувачам легко знаходити потрібні товари за різними критеріями, такими як ціна, розмір, колір та інші параметри.

Задача створення веб-порталу для підбору військової амуніції є досить комплексною, вона потребує створення якісної та надійної back-end частини, яка буде відповідати функціональності, що планується бути реалізованою в проекті, та зручного інтерфейсу з яким і буде взаємодіяти користувач.

Основним завданням є створення повноцінного веб-порталу для підбору військової амуніції, тобто як back-end частину так і front-end, в якому користувач зможе отримати весь необхідний функціонал для підбору та замовлення товару.

Важливою частиною будь-якого інтернет-магазину є авторизація користувача. Такий функціонал створюється для ідентифікації користувача при замовленні товару та зберігання даних для майбутнього зручного відображення інформації про всі замовлення та можливих персоналізованих пропозицій. Звичайно авторизація не має бути обов’язковою, тобто щоб вона не була блокуючим фактором який не дозоляє переглядати вміст сервісу. Це дуже важливий момент для утримання клієнтів, щоб користувач не витрачаючи часу міг переглянути асортимент та інформацію про магазин і тоді вже за потреби зареєструватись.

При реалізації реєстрації користувача слід приділити велику увагу на шифрування чутливих даних користувачів, наприклад паролів, щоб навіть якщо базу даних взламають, то зловмисники не зможуть використати вкрадені дані.

Головна функція таких веб-порталів – це перегляд інформації про товар, тому має бути зроблено все для зручності цього процесу. Для користувачів, які приблизно розуміють, який саме товар вони шукають буде зроблено сторінку з категоріями товарів. Користувач зможе ознайомитись які категорії є в магазині та перейти на потрібну, щоб переглянути товари по цій категорії. Для користувачів, які не впевнені що саме їм потрібно, буде зроблена сторінка зі всіма товарами, які є у веб-порталі.

Також не менш важливим функціоналом, який має бути реалізований у проекті це пошук та фільтрація. Такий функціонал значно спростить та пришвидшить підбір потрібного товару. Пошук буде зручним для людей, які шукають якийсь конкретний елемент військової амуніції, знаючи бренд або назву відповідного товару. Якщо в користувача є певні вподобання по кольору, потрібен конкретний розмір або щоб товар був визначеної ціни, фільтрація товарів буде незамінним помічником. Ці дві функції також повинні доповнювати одна одну, щоб була можливість використовувати їх одночасно.

При переході з сторінки товарів на вибраний продукт, користувач має побачити всю необхідну інформацію, а саме ціну, розміри та кольори які є в наявності.

Дуже зручним функціоналом, який має бути реалізований в будь-якому схожому веб-порталі, зберігання товару до обраного.

Для зручного адміністрування веб-порталу буде створена адмін-панель. Вона надасть можливість додавати товари та всю інформацію про них, опрацьовувати замовлення, а саме переглядати дані про замовлення, позначати чи доставлений товар. Також потрібно зробити можливість переглядати інформацію про користувачів. Для аналізу ефективності функціонування веб-порталу можна зробити графік, на якому буде зображена статистика продажів за певний час та графік, який буде зображати частку категорій від загальної кількості товару.

Робота складається з повного опису поставлених завдань, аналізу проблем та особливостей створення веб-порталів і аналізу схожих існуючих сервісів, дослідження засобів розробки веб-застосунків, опису основних елементів програмної реалізації системи та огляду роботи користувача з веб-порталом.

Дипломна робота виконана на 61 сторінці, містить 53 ілюстрації, 18 джерел в переліку посилань.

1 ЗАДАЧА СТВОРЕННЯ ВЕБ-ПОРТАЛУ ПІДБОРУ ВІЙСЬКОВОЇ АМУНІЦІЇ

Метою дипломної роботи є розробка веб-порталу підбору військової амуніції використовуючи сучасні технології для розробки.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1) проаналізувати існуючі веб-сервіси підбору військової амуніції для аналізу переваг та недоліків, які будуть враховані при розробці;

2) розробити архітектуру та структуру веб-порталу підбору військової амуніції;

3) розробити веб-портал для підбору військової амуніції, що виконуватиме наступні функції:

- авторизація користувача;
- відображення товарів;
- можливість додавання товарів у кошик та до обраного;
- оформлення замовлення;
- сторінки адміністратора;

4) провести комплексне тестування розробленого веб-порталу.

Продукт має бути створений зважаючи на потреби користувачів та враховуючи переваги і недоліки аналогічних сервісів. Портал має надати змогу зручно розміщувати товари на платформі, тримати під контролем весь шлях оформлення замовлення, як зі сторони клієнта, так і зі сторони адміністратора магазину. Важливим є забезпечення приємного досвіду користувача клієнтів, а саме зручність підбору товарів завдяки впровадженню пошуку та фільтрації.

Особлива увага має бути приділена можливості масштабування веб-порталу в майбутньому. Для цього будуть використані стандарти написання коду на вибраних технологіях та найкращі практики створення веб-сервісів.

Потенційними користувачами веб-порталу є військовослужбовці, волонтерські організації, а також приватні особи, які мають потребу у високоякісному військовому спорядженні.

2 АНАЛІЗ ІСНУЮЧИХ ВЕБ-ПОРТАЛІВ ПІДБОРУ ВІЙСЬКОВОЇ АМУНІЦІЇ

Для створення ефективного та зручного веб-порталу підбору військової амуніції потрібно провести дослідження, визначити потреби цільових користувачів та проаналізувати існуючі подібні сервіси. Це важливо для розуміння, які вони мають переваги щоб можливо додати такий функціонал в свій проект в майбутньому та вияснити недоліки, які можуть стримувати розвиток веб-порталу та незадовільняти потреби користувачів [1].

2.1 Аналіз потреб користувачів

Аналіз потреб користувачів є одним з найважливіших етапів розробки веб-порталу, оскільки він дозволяє створити продукт, який максимально відповідає очікуванням та вимогам цільової аудиторії.

Для початку потрібно зрозуміти хто буде користуватись створеним сервісом для підбору військової амуніції, тобто кому може бути корисним такий веб-портал. Цільова аудиторія може включати:

– військовослужбовців, а саме людей які безпосередньо беруть участь у бойових діях та потребують якісного спорядження;

– волонтерські організації, які займаються волонтерською діяльністю і закуповують необхідні засоби на потреби військових та відправляють в зону бойових дій;

– державні установи, що займаються закупівлею військового спорядження для потреб армії;

– цивільні люди, яким подобається тактичний одяг для різної діяльності, наприклад риболовля, полювання, походи в гори та інші види активного відпочинку.

Визначивши потенційних користувачів веб-порталу потрібно зрозуміти, які саме вимоги до програмного забезпечення в них є. Цей процес можна здійснити за допомогою різних методів, таких як:

– проведення опитувань серед потенційних користувачів, що дозволяє отримувати зворотній звя’зок щодо їх вподобань, потреб та очікувань;

– проведення великих інтерв’ю з окремими користувачами для отримання більш глибокої інформації про потреби та проблеми з якими вони стикаються при підборі військової амуніції. Цим самим можна зрозуміти чого не вистачає в конкурентів та недоліки існуючих рішень на ринку;

– вивчення існуючих веб-порталів для підбору військової амуніції щоб визначити, які функціональні можливості вони пропонують, які мають слабкі та сильні сторони. Це допомагає зрозуміти що потрібно покращити, на чому зробити особливий акцент та які функції є важливими для користувачів.

Провівши опитування серед своїх знайомих військовослужбовців, було вияснено їхні вподобання. Всі опитані відмітили наступні фактори, які найбільше впливають на досвід користування такими сервісами:

1. Зручність навігації та інтерфейсу. Будь-яка людина з опитаних не хоче задумуватись, що їй потрібно натиснути щоб потрапити на якусь сторінку. Відмічено що дуже складним інтерфейсом, хоча функціональним неприємно користуватись. Інтерфейс повинен бути інтуїтивно зрозумілий і достатньо простим, щоб можна було швидко отримати доступ до категорій товарів, так як це основний функціонал. Зручна система пошуку та фільтрації є обов’язковою для користувачів.

2. Детальна інформація про товар. Користувач повинен мати доступ до детального опису товару, включаючи інформацію про наявність, колір, ціну та розміри.

3. Швидкість роботи сервісу. Було визначено, що для користувачів дуже важливою є швидкодія сервісу. Нікому не подобається чекати декілька секунд поки провантажиться сторінка на сайті.

4. Якісна підтримка користувачів. Також опитані військовослужбовці відмітили, що в багатьох веб-порталах для підбору амуніції не вистачає достатньої ефективної підтримки клієнтів. В багатьох сервісах є номер телефону, куди можна зателефонувати щоб отримати необхідну інформацію, але було б зручно якби ще

був розділ часто задаваних питань (FAQ), так як не завжди є бажання та час щоб телефонувати по якомусь незначному запитанню.

5. Наявність локалізації. Декілька іноземних військовослужбовців повідомили що в більшості українських веб-порталах немає локалізації, а саме немає можливості перемкнути мову сервісу на англійську, що значно ускладнює користування для іноземних клієнтів.

Отже, послухавши думку цільової аудиторії, було зроблено висновки, що саме обов'язково потрібно реалізувати в проекті для залучення клієнтів. Але незважаючи на це, після запуску веб-порталу важливо продовжувати підтримувати зворотній зв'язок з користувачами, щоб швидко підлаштовуватись під потреби та постійно покращувати систему.

Таким чином, аналіз потреб користувачів дозволяє створити веб-портал, який максимально відповідає сучасним вимогам цільової аудиторії, забезпечуючи високу якість обслуговування та задоволення потреб користувачів.

2.2 Аналіз подібних веб-порталів підбору амуніції

Для того щоб зробити ефективний та зручний сервіс потрібно детально проаналізувати існуючі веб-портали для підбору військової амуніції. Це дозволить визначити сильні та слабкі сторони конкурентів, взяти на озброєння кращі практики та функціональності і уникати можливих помилок. У цьому підрозділі розглянемо декілька українських веб-порталів, які спеціалізуються на продажі військової амуніції та займають провідні позиції на ринку.

2.2.1 Інтернет-магазин PROF1GROUP

Одним з сервісів підбору військової амуніції є інтернет-магазин PROF1GROUP [2] (рисунок 2.1). Для аналізу вибраний саме він, так як неодноразово була інформація що багато військових підбирають собі військове спорядження там і також його достатньо легко знайти в пошукових системах.

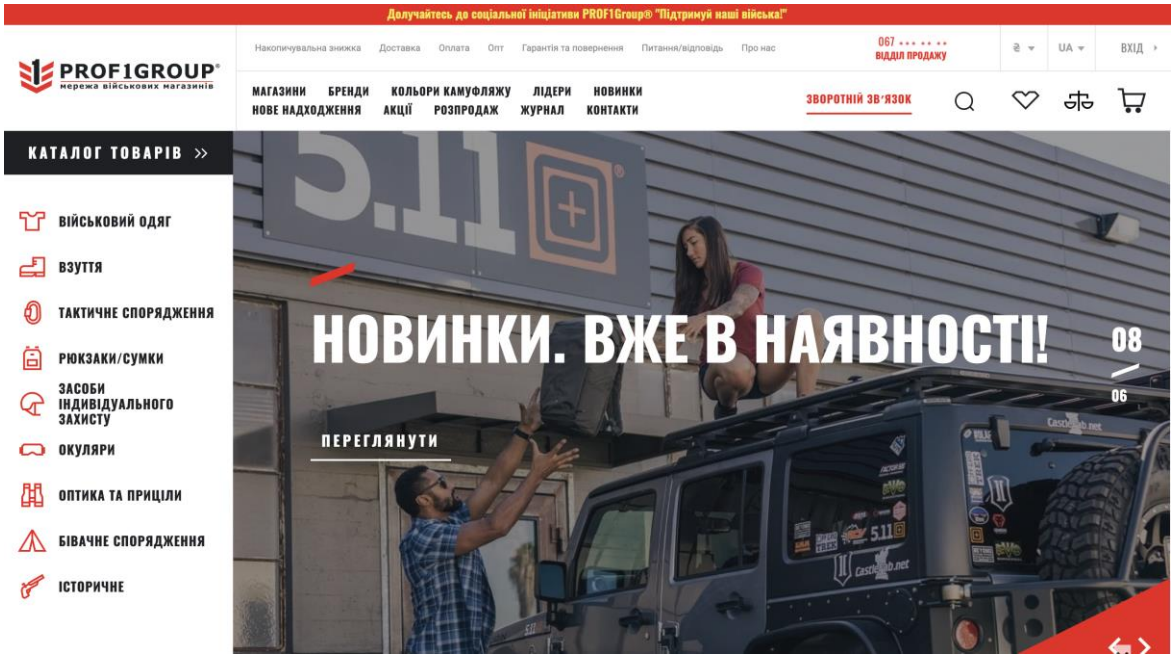


Рисунок 2.1 — Головна сторінка PROF1GROUP

Веб-портал має достатньо багато переваг, що робить його непоганим сервісом. Основними перевагами є:

– великий асортимент товару. Асортимент включає військове спорядження, різноманітні аптечки, намети, гамаки та навіть приціли і біноклі;

– багато функціональності. У інтернет-магазині є безліч різноманітних функцій, які роблять підбір спорядження легким та зручним, такі як пошук, можливість додавання товарів у збережене, порівняння товарів, авторизація;

– на сайті реалізована локалізація, що робить зручним використання сервісу іноземним користувачам;

– створене розділення товарів по категоріям;

– є багато корисних сторінок, наприклад сторінка з інформацією про доставку, знижки, акції та часто задавані питання;

– підтримка різних платіжних систем.

Незважаючи на всі ці переваги, інтернет-магазин має також і недоліки. Основним недоліком є перевантаженість інформацією.

Для того, щоб отримати детальний аналіз сервісу скористаємось інструментом Lighthouse, який автоматично проаналізує веб-сторінки з метою оцінення продуктивності, доступності, оптимізації для пошукових систем та

прогресивності [3]. Виконаємо аналіз головної сторінки, оскільки це основна сторінка і вона має бути створена найкраще для зручності користувача.

Отримуємо результат зображений на рисунку 2.2.

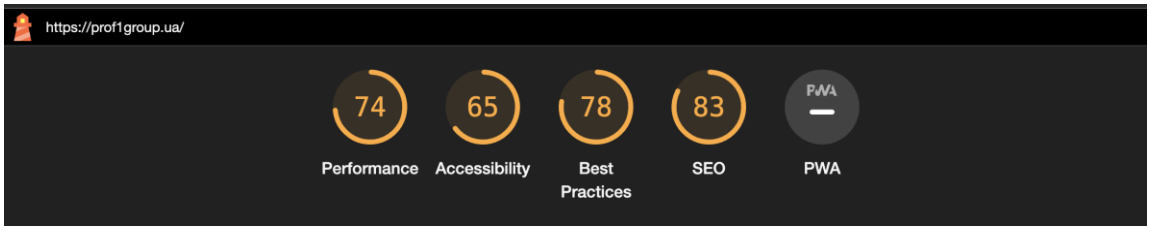


Рисунок 2.2 — Значення метрик звіту Lighthouse

З результату звіту можна побачити, що показники середні відповідно до градації Lighthouse, рисунок 2.3.

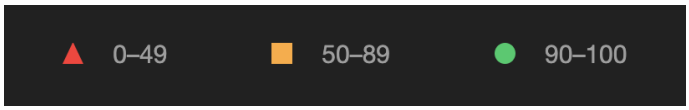


Рисунок 2.3 — Градація метрик

Середня оцінка продуктивності пов’язана з часом відображення найбільшого контенту на сторінці, а саме 3.4 секунди, рисунок 2.4.

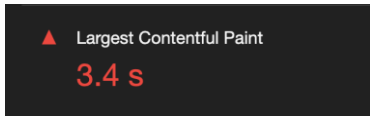


Рисунок 2.4 — Час відображення найбільшого контенту

Ця метрика важлива для користувачів, оскільки вона відображає, який час буде очікувати користувач щоб побачити основний контент сторінки.

Метрика Speed Index в Lighthouse виявилась теж не високою, а саме 2.5 секунди (рисунок 2.5).

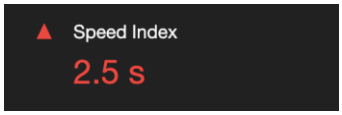


Рисунок 2.5 — Індекс швидкості завантаження

Вона відображає швидкість відображення контенту на сторінці. Це може суттєво негативно вплинути на задоволення від користування інтернет-магазином.

Дивлячись на метрику Best Practices можна сказати що розробники використовували хороші практики розробки, але все ж значення є не найвищим.

SEO оптимізація виконана нормально, але залишається на рівні вище середнього.

Щоб зрозуміти чому саме такі вийшли оцінки можемо визначити на яких технологіях написаний веб-портал за допомогою сервісу “Built With”. Це сервіс, який надає інформацію про технології, які використовувались при розробці веб-сайту, наприклад мови програмування, фреймворки. Сервіс дуже корисний для дослідження ринку, вибору правильних технологій для розробки та аналізу тенденцій технологій.

Щоб отримати інформацію введемо адресу веб-сайту який нас цікавить, рисунок 2.6.

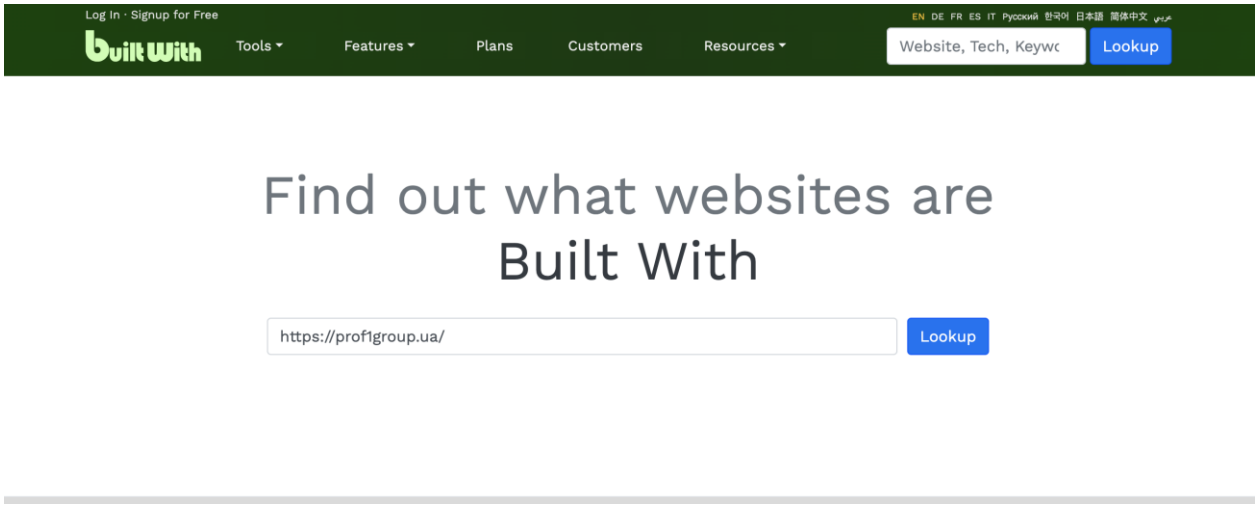


Рисунок 2.6 — Головна сторінка сервісу “Built With”

Після аналізу сервіс видає за розділами технології, які використовуються на сайті. Найбільше цікавить розділ з JavaScript бібліотеками, так як їхній вибір напряму впливає на якість розробки (рисунок 2.7).

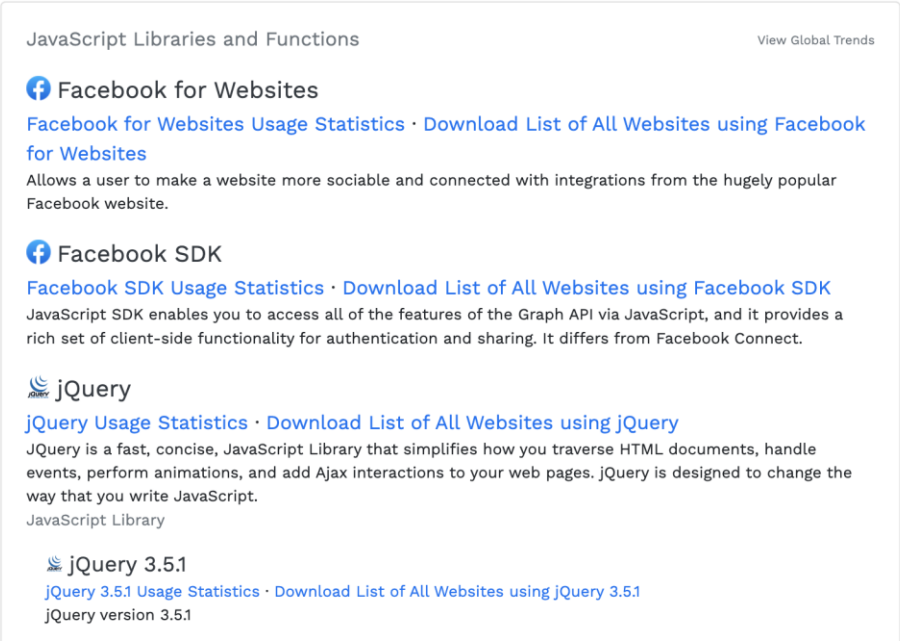


Рисунок 2.7 — Результат аналізу сервісу “Built With”, розділ з JavaScript бібліотеками

Можна побачити, що основною бібліотекою, яка була використана є jQuery. В розробці сучасних веб-сайтів jQuery є не дуже вдалим вибором, оскільки є застарілою та має значну кількість недоліків. Такими недоліками є:

– великий розмір, що може впливати на час завантаження сторінок;

– залежність від DOM-операцій. Оскільки jQuery працює напряму з DOM-деревом це також впливає на повільну реакцію на зміни сторінок;

– немає зручної можливості реалізувати SSR (Server-Side Rendering) навідміну від сучасних фреймворків як Next.js та Nuxt.js;

– недостатня підтримка нових стандартів.

Підводячи підсумки, інтернет-магазин PROF1GROUP вцілому справляється зі своїм завданням і має достатню кількість переваг, які можна використати в майбутньому проекті, такі як локалізація, корисні інформаційні сторінки, пошук. Але в той самий час містить недоліки, а саме не дуже хороша продуктивність, SEO оптимізація та недостатнє використання найкращих практик, що спричинено неправильно підібраними технологіями. А також проблеми з перевантаженням інформацією та зайвим функціоналом.

2.2.2 Інтернет-магазин Tactical Gear

Розглянемо ще один інтернет-магазин військової амуніції, Tactical Gear [4], який іноді використовують військові через невисоку ціну товарів і також його неважко знайти в пошуку. Головна сторінка зображена на рисунку 2.8.

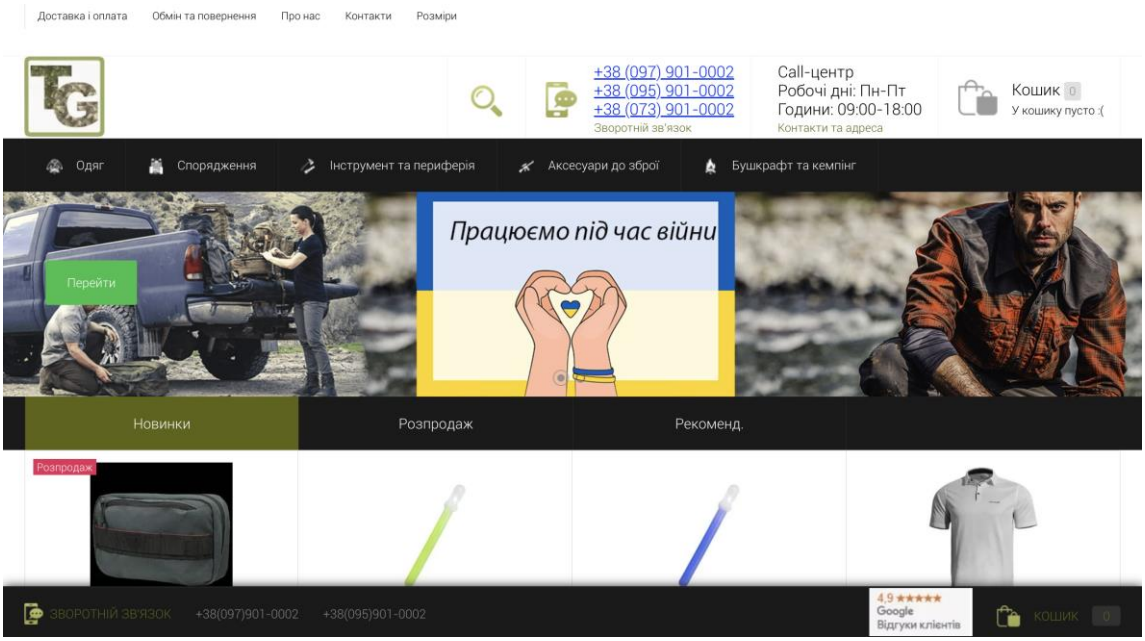


Рисунок 2.8 — Головна сторінка Tactical Gear

Цей сервіс зацікавлює своєю простотою у користуванні. Наприклад, відразу при переході на сайт відображені категорії товарів. Також можна переглянути новинки, розпродаж та рекомендації. Тобто немає нічого зайвого що б могло відштовхнути та забрати увагу. На перший погляд це дуже простий, але в той самий час функціональний магазин. Приємно, що є мобільна адаптація сайту, а це означає, що користувачам буде зручно ним користуватись як на ноутбуках та комп'ютерах, так і на мобільних пристроях.

Недоліки на які відразу можна звернути увагу – це дизайн самого веб-порталу. Він виглядає не дуже сучасно і в ньому немає чогось особливого щоб могло зацікавити користувача, він засильно простий. Також у верстці використовуються застарілі іконки (рисунок 2.9), а це означає що над дизайном сервісу погано працювали.

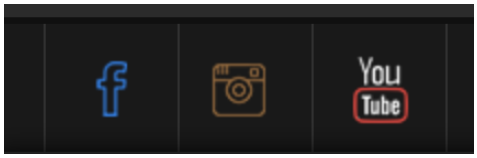


Рисунок 2.9 — Застарілі іконки

Якщо перейти на певну категорію товарів, щоб легше знайти потрібний товар є фільтр по багатьом параметрам (рисунок 2.10), що дуже зручно.

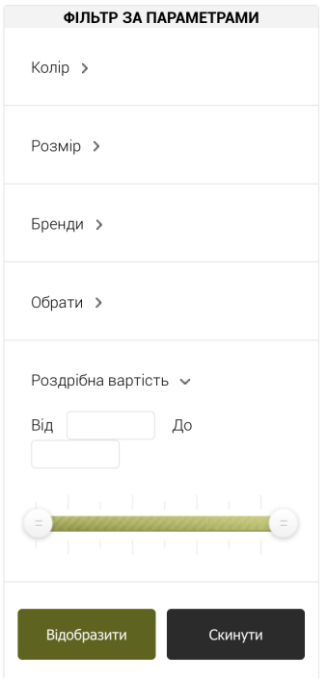


Рисунок 2.10 — Фільтр товарів

Але немає пошуку по товарам, тобто якщо користувач зайшов у магазин щоб подивитись якийсь конкретний товар, то йому прийдеться витрачати час на вибір фільтрів та гортання сторінок з результатами. Якщо товарів у магазині багато, то це може спричинити незручності та погіршити досвід користування інтернет-магазином у клієнтів. Потенційний клієнт може не знайти потрібний товар через деякий час і просто покинути магазин, хоча товар може бути в наявності.

Недоліком веб-порталу є відсутність локалізації, тобто він не підлаштований для іноземних користувачів. Це не є значним недоліком, якщо сайт націлений тільки на українську аудиторію, але все ж таки, якщо іноземний користувач зайде

в магазин, то з великою ймовірністю він його залишить та перейде на той веб-портал де є локалізація.

Виконавши аналіз через Lighthouse, отримали наступні результати, показані на рисунку 2.11.

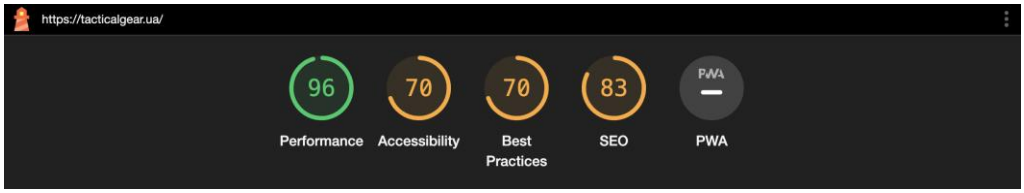


Рисунок 2.11 — Звіт Lighthouse головної сторінки Tactical Gear

Через свою простоту реалізації веб-портал має дуже хорошу продуктивність, а це означає що він працює швидко і без затримок. Інші ж метрики знаходяться на середньому рівні.

Проаналізувавши даний інтернет-магазин можна зробити висновок, що простота є важливим аспектом для подібних сервісів. Цим самим збільшується продуктивність та покращується досвід роботи користувача з системою, але не потрібно забувати про якісний, сучасний та гарний інтерфейс.

3 ЗАСОБИ РОЗРОБКИ ВЕБ-ПОРТАЛУ

Під час створення будь-якого програмного продукту постає питання у виборі засобів розробки, а саме які технології та інструменти будуть використані для реалізації проекту. Цей етап є дуже важливим, так як від засобів розробки залежить швидкість самої розробки, забезпечення безпеки даних, продуктивність створеної системи, підтримування та масштабування проекту.

У цьому розділі будуть розглянуті технології, які використовувались при розробці веб-порталу.

Перш за все, під час вибору засобів розробки важливо проаналізувати поточний ринок технологій. Популярні фреймворки та бібліотеки зазвичай мають велику спільноту користувачів, що створює численні переваги. Велика кількість активних користувачів зазвичай означає що багато питань та проблем вже були вирішені і відповіді на них можна знайти на різних веб-ресурсах таких як Stack Overflow або відповідне відео на YouTube. Також якщо технологія дуже популярна то це свідчить про наявності різноманітних навчальних матеріалів, що значно спрощує процес її освоєння. Для початку переглянемо рейтинг веб-фреймворків.

Не менш важливим фактором при виборі технологій є сучасність. Сучасні технології зазвичай забезпечують кращу продуктивність, безпеку та гнучкість. Обравши таку технологію можна бути впевненим що вона буде ще довгий час підтримуватись та розвиватись, будуть з’являтись нові функції та особливості. Крім того, нові технології часто мають поліпшену архітектуру та функціонал, що дозволяє розробникам витрачаючи менше часу та зусиль створювати більш надійні та ефективні додатки.

Для нашого проекту було обрано Next.js для написання front-end частини та back-end. Фреймворк є сучасним та має дуже багато переваг, що робить програмні продукти на основі нього швидкими, SEO-оптимізованими та масштабованими [5].

Мовою програмування для розробки було обрано TypeScript через його статичну типізацію та підвищену надійність, що полегшує його масштабування та підтримку в майбутньому.

Для зберігання даних було обрано MongoDB, оскільки ця база даних забезпечує високу продуктивність та гнучкість. Також вона дуже легко інтегрується з Next.js.

Для управління станом застосунку використовується Zustand, який є сучасною та легковаговою бібліотекою, яка забезпечує простий і ефективний спосіб керування станом у React-застосунках. Таке рішення робить програмний продукт простим, зрозумілим, але водночас високопродуктивним.

Ще однією важливою технологією, використаною у проекті, є Tailwind CSS. Це CSS фреймворк, який дозволяє швидко створювати сучасні та адаптивні інтерфейси. Завдяки своїй гнучкості та зручному використанню він значно прискорює розробку, дозволяючи не витрачати час на написання архітектури стилів, а зосередитись на функціональності.

Також для швидкої розробки багаторазових компонентів використовувалась сучасна колекція компонентів shadcn. Вона надає компоненти, які легко стилізуються і повністю налаштовані для використання. Такий підхід дуже зручний, так як не потрібно задумуватись над тим як реалізувати функціонал певного елементу застосунка, а лише стилізувати за бажанням.

3.1 Фреймворк Next.js

Для початку щоб обрати основну технологію для розробки було проаналізовано рейтинг популярності веб-фреймворків та бібліотек, рисунок 3.1.

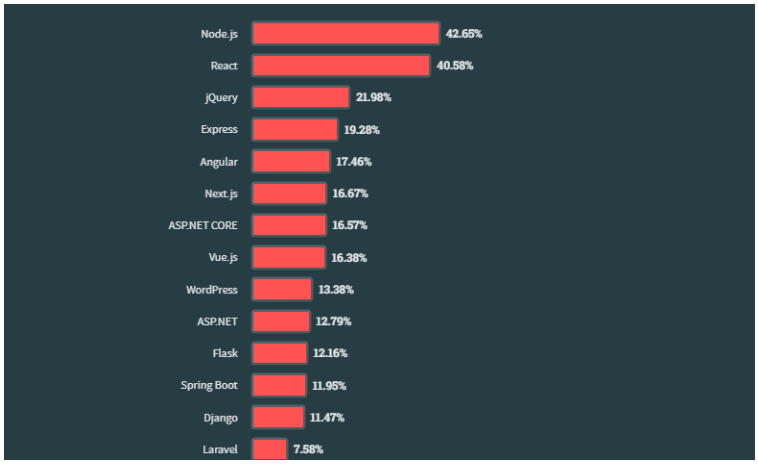


Рисунок 3.1 — Рейтинг веб-фреймворків та бібліотек

Бачимо що React посідає друге місце за популярністю. React – це бібліотека для розробки інтерфейсу користувача, яка дозволяє створювати багаторазові компоненти, які можна комбінувати, створюючи складні інтерфейси.

React має багато особливостей, які роблять розробку зручною. Бібліотека використовує дуже сучасний підхід для розробки інтерфейсів, а саме компонентний. Такий підхід все частіше використовують, навіть при розробці дизайну інтерфейсів. React використовує концепцію віртуального DOM для підвищення продуктивності [6]. Така концепція означає, що всі зміни в компонентах спочатку застосовуються до віртуального DOM, потім віртуальний та реальний DOM порівнюються для визначення змін і далі набір змін застосовуються вже до реального DOM. Завдяки використання такого підходу React забезпечує так звану “реактивність”, а саме високу продуктивність і швидкість оновлення інтерфейсу.

Для створення front-end частини був обраний фреймворк Next.js, який працює на основі бібліотеки React. Це фреймворк, який підтримує серверний рендеринг та статичну генерацію сторінок.

Next.js має ряд переваг, які вирізняють його від конкурентів. Основною особливістю, яка притягує розробників використовувати цей фреймворк є серверний рендеринг (SSR). Це процес генерації сторінок на сервері перед відправкою коду на сторону клієнта. Під час звичайного підходу, рендеринг відбувається на клієнті за допомогою JS після завантаження сторінки [7]. Таким чином SSR забезпечує швидкодію завантаження сторінок, а це позитивно впливає на продуктивність всього сервісу, що є критичним для інтернет-магазину. Також не менш важливою перевагою серверного рендерингу є SEO-оптимізація. Завдяки тому що на клієнт приходить вже готовий HTML код сторінки, він набагато краще індексується пошуковими системами та роботами. Next дозволяє реалізовувати серверний рендеринг з “коробки” на відміну від багатьох інших бібліотек та фреймворків в яких для такого функціоналу потрібно робити різні налаштування та використовувати сторонні бібліотеки, що значно ускладнює розробку та майбутню підтримку сервісу.

Next.js має гібридний підхід до рендерингу сторінок, а саме окрім рендерингу на сервері є можливість статичної генерації сторінок (SSG), що також покращує продуктивність всього сервісу та зменшує навантаження на сервер, завдяки тому що сторінки створюються під час побудови застосунка, а не на кожен запит користувача, тим самим розвантажуючи сервер [8]. Такі сторінки зберігаються як статичні файли і тому завантажуються миттєво.

Одним з найбільших переваг фреймворку Next.js, яка відіграла важливу роль під час вибору цієї технології для реалізації мети роботи, є вбудований роутинг. Він дозволяє автоматично створювати маршрути на сайті на основі файлової структури проекту без додаткових бібліотек. Це спрощує управління маршрутами та забезпечує гнучкість у створенні динамічних сторінок. Якщо б розробка відбувалась тільки за допомогою бібліотеки React, то для роутингу потрібно було б використовувати сторонні бібліотеки.

Також фреймворк має вбудовані автоматичні механізми оптимізації проекту, а саме оптимізації зображень, шрифтів та ліниве завантаження, що теж позитивно впливає на продуктивність створеного сервісу.

Next.js це зручне та сучасне рішення для розробки front-end частини веб-сайтів, яке не потребує додаткових бібліотек та налаштувань для основних етапів розробки.

Для написання серверної частини веб-порталу було вирішено використовувати теж Next.js, оскільки це не тільки фреймворк для написання інтерфейсів, а і є можливість розробки повноцінного API. Недоліком цього засобу для великих проектів є складність масштабування за рахунок нагромадження проекту. Якщо дивитись з боку невеликих та середніх проектів, таких як розроблений веб-портал, то це дуже зручне рішення. Перш за все не потрібно створювати окремі проекти для front-end та back-end частини. Завдяки тому, що все зберігається в одному проекті, з’являється легкість взаємодії між частинами. Також для початківців, дуже зручно що не потрібно опановувати декілька різних технологій для створення повноцінного проекту. Next.js пропонує простий і зрозумілий спосіб реалізації серверних функцій без необхідності налаштувань додаткових конфігурацій. Також фреймворк легко інтегрується з різними базами

даних такими як, MongoDB, PosgreSQL та іншими, що дозволяє створювати повноцінні бекенд-системи. API роутинг Next підтримує всі основні HTTP методи, а саме GET, POST, PUT, DELETE, а це надає можливість легко створювати RESTful сервіси.

Отже Next.js є потужним фреймворком, який не тільки спрощує розробку front-end, але і надає можливості та багато переваг для створення back-end. Завдяки серверному рендерингу, вбудованому роутингу, оптимізації, простоті інтеграції з базами даних та іншими технологіями це дуже хороше рішення, яке дозволяє створювати сучасні, швидкі та надійні веб-додатки.

3.2 Мова програмування Typescript

Мовою для розробки було обрано TypeScript. Це модернізована мова програмування JS в яку додали типізацію. При розробці більшості великих проектів використовують саме TS, оскільки він має ряд переваг над звичайним JavaScript.

Основною перевагою чому була обрана ця мова є краща надійність. Статична типізація дозволяє виявляти помилки на етапі компіляції, що набагато зменшує ризики написання непрацюючого коду, через звичайні помилки [9]. При написанні коду на звичайному JavaScript використовуючи бібліотеку React частою проблемою було передавання неправильних props до компонентів і це не швидко можна помітити.

Не менш важливою перевагою є краща масштабованість. Проект написаний на TypeScript набагато легше масштабувати та підтримувати за рахунок хорошої організації коду і кращого його розуміння навіть через довгий час.

Крім того, TS забезпечує покращену читабельність коду, завдяки явній декларації типів для змінних, об’єктів, параметрів функцій. При розробці проекту на TS коли потрібно приймати дані з бекенду, спочатку описуються всі типи. Після того при зверненні до отриманих даних, середовище розробки підказує що саме міститься там, якщо це об’єкт, то які властивості містяться в цьому об’єкті та які методи можна використовувати для них. Також використовуючи TypeScript, IDE може надавати більше доповнень вашому коду та інших підказок.

Отже, для написання якісного, сучасного коду з найкращими практиками, вибір TypeScript є оптимальним, він дозволить підняти якість, безпеку та продуктивність розробки.

3.3 База даних MongoDB

Для збереження даних про товари, користувачів, замовлення була вибрана база даних MongoDB. Цей вибір обумовлений її гнучкістю та сучасними можливостями, які відповідають потребам проекту.

Це нереляційна база даних, яка на відміну від звичайних баз даних, таких як MySQL та PostgreSQL, зберігає дані не в таблицях, а використовує для цього схему документів, що дозволяє зберігати дані різних типів у їхньому звичайному вигляді. Такі бази даних дуже гнучкі та адаптуються до змін у структурі даних. Вони мають хорошу продуктивність, яку забезпечує висока швидкість обробки запитів та доступу до даних [10].

Однією з ключових переваг MongoDB є можливість горизонтального масштабування. Це означає, що базу даних можна розширювати шляхом додавання нових серверів, а це дозволяє обробляти великі обсяги даних та велику кількість користувачів.

Особливо зручно використовувати MongoDB в проекті написаному на TypeScript, завдяки об’єктно-документному менеджеру для Node.js - Mongoose, який забезпечує роботу з MongoDB. Він полегшує взаємодію з базою даних та надає можливості для визначення схем, валідації та створення моделей. Синтаксис Mongoose дуже простий, тому ідеально підійде користувачам, які тільки починають займатись back-end розробкою.

Таким чином, використання MongoDB в проекті дозволяє ефективно та зручно керувати великим обсягом даних, при цьому забезпечуючи високу продуктивність і надійну роботу програмного продукту, що дуже важливо для веб-порталу підбору військової амуніції.

3.4 Бібліотека Zustand

Для управлінням станом у проекті було обрано бібліотеку Zustand. Вибір обумовлений її простотою та гнучкістю, що робить її хорошим рішенням у веб-застосунках написаних на Next.js.

Zustand забезпечує легке та зрозуміле управління станом без зайвої складності на основі хуків. Однією з особливостей синтаксису є його схожість з концепцією інкапсуляції. Він дозволяє створювати стан та функції для його зміни у межах окремих модулів, приховуючи внутрішні деталі реалізації від зовнішнього коду. Такі модулі зазвичай зберігаються в окремих файлах і називаються сторами.

На відміну від інших бібліотек для управління станом, таких як Redux не вимагає створення великої кількості шаблонного коду та налаштувань, а дозволяє зосередитись на розробці.

Гнучкість Zustand дозволяє легко його інтегрувати у будь-який проект написаний на основі React, так як вона дає можливість описувати як прості, так і складні сценарії управління станом.

Також бібліотека відзначається високою продуктивністю, оскільки використовує оптимізовані механізми оновлення стану, що забезпечує швидке та ефективне реагування на зміни стану [11]. Це дуже важливо для програмних продуктів у яких продуктивність є критичним фактором.

Отже, використання бібліотеки Zustand та її функцій у проекті, забезпечує зручне та ефективне керуванням станом, що сприяє швидкій розробці та легшій підтримці веб-застосунку в майбутньому. Такий підхід дозволяє зосередитись на розробці та впровадженні нового функціоналу, не витрачаючи багато часу на налаштування та прописування додаткового коду для управління станом.

3.5 Фреймворк Tailwind CSS

Для стилізації компонентів та всього застосунку був обраний CSS-фреймворк Tailwind CSS. Він відомий своїм підходом до утилітарного стилізування, що забезпечує високу гнучкість та швидкість розробки.

Основна перевага фреймворку полягає в його утилітарному підході. Використовуючи його, для стилізації елементів не потрібно створювати CSS класи з набором властивостей та придумувати для них назви. Tailwind надає різноманітні класи які по функціоналу відповідають певним CSS властивостям, які також можна комбінувати. Це дозволяє значно пришвидшити розробку, зменшити кількість файлів проекту та коду. Фреймворк генерує мінімальний обсяг CSS коду, що покращує швидкість завантаження сторінок та зменшує використання пам’яті [12].

Таким чином, використання Tailwind CSS зменшує кількість коду, забезпечує швидку та ефективну розробку, продуктивних веб-застосунків. Він є ідеальним рішенням для тих хто хоче створювати цікаві та красиві інтерфейси з мінімальними зусиллями.

3.6 Колекція компонентів shadcn/ui

Shadcn – це колекція повторно використовуваних компонентів. Як наголошують самі розробники, це не бібліотека, так як не потрібно встановлювати якісь залежності. Достатньо просто скопіювати код потрібного компоненту собі в проект, кастомізувати і використовувати. Ця колекція вирізняється своєю здатністю інтегруватись в будь-які інтерфейси, оскільки компоненти не містять специфічних стилів і вони дуже легко стилізуються за допомогою Tailwind CSS.

Shadcn надає широкій набір готових компонентів, які можна адаптувати під потреби проекту. Кожен компонент ретельно розроблений з урахуванням найкращих практик дизайну та UX, що забезпечує високу якість створеного інтерфейсу [13]. Також завдяки тому що компоненти дуже оптимізовані, вони забезпечують швидке завантаження сторінок і загальну продуктивність веб-порталу.

Отже, для створення сучасних інтерактивних інтерфейсів будь-якої складності компоненти shadcn підходять чудово, завдяки своїй гнучкості, оптимізованості та легкій взаємодії.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ПОРТАЛУ

У цьому розділі описується програмна реалізація веб-порталу для підбору військової амуніції. Розділ буде включати функціональну декомпозицію системи, а саме діаграму прецедентів, для кращого розуміння хто буде діячами в системі та які дії вони зможуть виконувати. Буде продемонстрована структура бази даних, тобто її складові та взаємозв'язки між сутностями. Детально розглянемо ініціалізацію проекту Next.js та налаштування середовища розробки. Особлива увага приділяється розробці основних функціональних компонентів системи як back-end частини, так і front-end.

4.1 Архітектура веб-порталу

Архітектура веб-порталу базується на клієнт-серверній моделі, яка забезпечує розділення функцій між клієнтською частиною (front-end) та серверною (back-end). Це дозволяє створити гнучку та масштабовану систему.

Клієнтська частина відповідає за взаємодію користувача з системою, іншими словами це інтерфейс. Основні функції клієнтської частини включають: відображення товарів, можливість додавання до кошика та до обраного, процес оформлення замовлення, перегляд профілю.

Серверна частина відповідає за обробку запитів від клієнтів, управління даними, взаємодія з базою даних. Також вона зазвичай керує бізнес-логікою системи. Основні функції серверної частини включають: управління товарами, а саме додавання, видалення та редагування даних в БД, управління користувачами в БД, створення замовлень. Можна сказати що більшість основних дій, які виконує користувач у веб-порталі використовують за собою серверну частину для маніпулювання базою даних. Back-end забезпечує безпеку завдяки механізмам автентифікації та авторизації користувачів. І також відповідає за захист даних при передачі між клієнтом і сервером використовуючи протоколи безпеки, наприклад HTTPS.

Клієнтська частина взаємодіє з серверною через HTTP-запити до API, що забезпечує обмін даними між front-end та back-end частинами сервісу. Наприклад, коли користувач додає товар, клієнт надсилає POST-запит на сервер, і він його обробляє, маніпулює з БД і повертає відповідь клієнту. Схема архітектури зображена на рисунку 4.1.

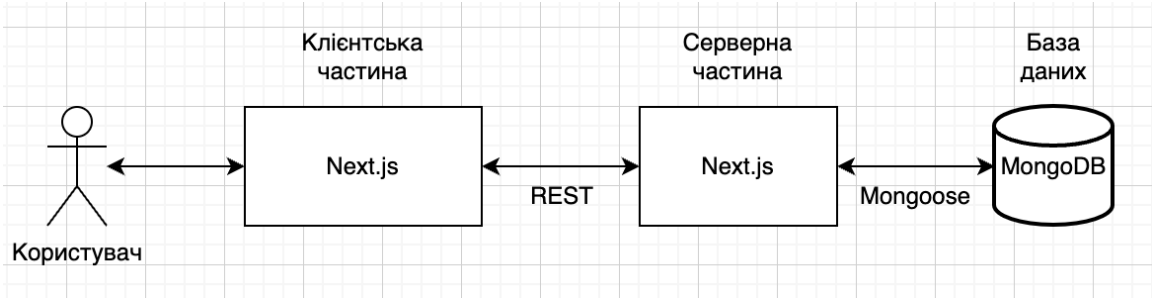


Рисунок 4.1 — Схема клієнт-серверної архітектури

Діаграма демонструє основні компоненти системи та напрямки взаємодії між ними. Користувач взаємодіє з клієнтською частиною, яка надсилає запити до серверу. Сервер звертається до бази даних за необхідною інформацією та повертає відповіді клієнтській частині.

Отже, така архітектура забезпечує чітке розділення відповідальності між front-end та back-end. Кожен компонент системи не залежить напряму один від іншого, що дозволяє легко масштабувати систему та підтримувати її.

4.2 Функціональна декомпозиція системи

Функціональна декомпозиція системи є важливим етапом у процесі розробки веб-порталу, вона дозволяє визначити функціональні можливості системи, ролі користувачів та їх взаємодію з системою. На рисунку 4.2 зображено діаграму прецедентів, яка демонструє функціональність веб-порталу для двох типів користувачів: адміністратора сервісу та клієнта.

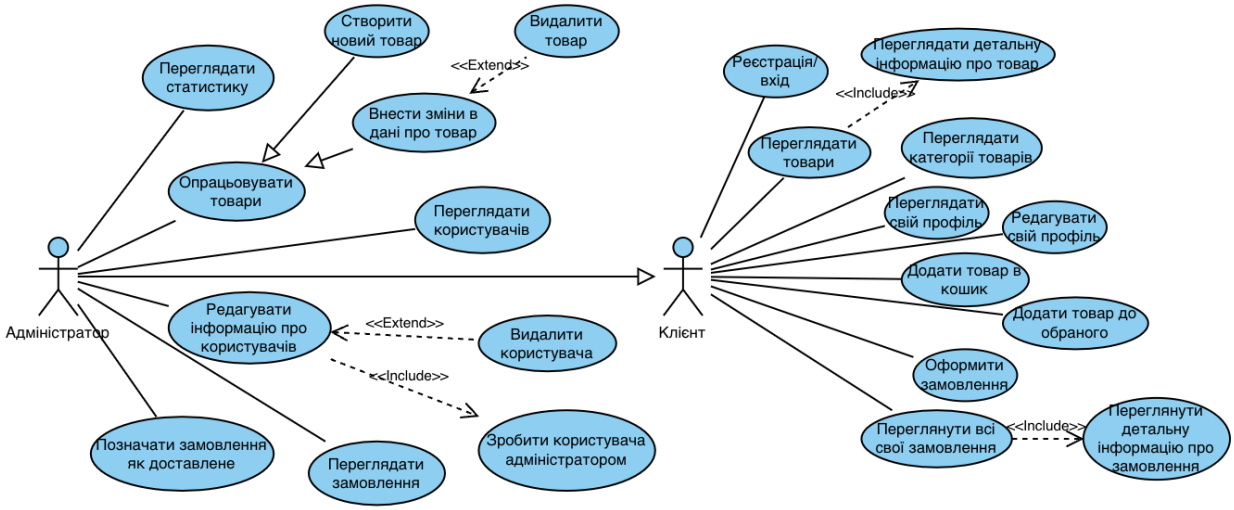


Рисунок 4.2 — Діаграма прецедентів

Адміністратор має широкий спектр можливостей для керування системою. Він може переглядати статистику, що включає аналіз активності на веб-порталі, такої як кількість замовлень, товарів, користувачів, графік продажів та діаграму кількості товарів по категоріям. Опрацьовування товарів включає створення нових товарів, внесення змін до інформації про існуючі товари та їх видалення. Ці функції дозволяють підтримувати актуальність та точність інформації про товари на веб-порталі. Адміністратор також має можливість переглядати користувачів, редагувати їх інформацію, видаляти користувачів з системи та надавати їм права адміністратора. Це забезпечує контроль над користувачами та їхніми правами доступу. Крім того, адміністратор може позначати замовлення як доставлені, що дозволяє відстежувати статус замовлень та забезпечувати своєчасну доставку товарів. Перегляд замовлень дає можливість адміністратору бачити всі зроблені замовлення та їх деталі. Також адміністратор може виконувати всі дії клієнта.

Клієнти мають доступ до функцій, що дозволяють їм взаємодіяти з веб-порталом для підбору та замовлення військової амуніції. Вони можуть реєструватися на веб-порталі, створюючи свій обліковий запис, та входити в систему для доступу до персоналізованих функцій. Перегляд товарів включає можливість переглядати список доступних товарів, отримувати детальну інформацію про кожен товар та переглядати товари за категоріями. Клієнти можуть додавати товари до кошика для подальшого замовлення або до списку обраного для

майбутніх покупок. Оформлення замовлень дозволяє клієнтам здійснювати покупки, а перегляд всіх своїх замовлень дає можливість бачити історію своїх замовлень та їх деталі. Вони також можуть переглядати та редагувати свій профіль, що дозволяє їм підтримувати актуальність своїх персональних даних.

Отже, функціональна декомпозиція системи та діаграма прецедентів допомагають повністю зрозуміти функціонал веб-сервісу та при розробці не пропустити важливі елементи системи.

4.3 Налаштування середовища розробки та ініціалізація проекту

Щоб розробка була зручною та легкою необхідно правильно підібрати середовище розробки, яке буде відповідати нашим потребам. На даний момент існує багато різноманітних редакторів коду та інтегрованих середовищ розробки, але на мою думку найкращим рішенням для веб-розробки є WebStorm. Це повноцінне IDE (Integrated Development Environment), яке має дуже багато різноманітних функцій що значно спрощує розробку та навігацію по проекті.

Обравши інструменти для роботи, можна ініціалізувати проект з початковими налаштуваннями. Оскільки основною технологією для розробки як front-end частини, так і back-end буде використовуватись Next.js, достатньо створити один проект. Для ініціалізації проекту розробник має можливість скористатись зручною командою “create-next-app”, яка після відповідей на запитання про налаштування проекту створить основну структуру проекту та встановить всі необхідні залежності для подальшої розробки. Процес ініціалізації проекту зображений на рисунку 4.3.

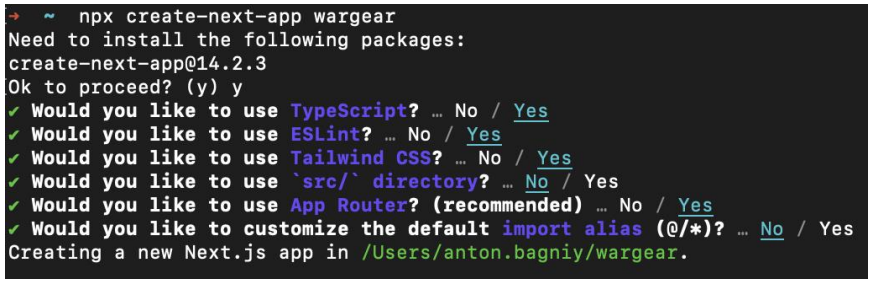


Рисунок 4.3 — Процес ініціалізації проекту

Коли процес створення завершиться, можемо перейти до проекту. Бачимо створені початкові файли, які потрібні для написання коду. Структура ініціалізованого проекту показана на рисунку 4.4.

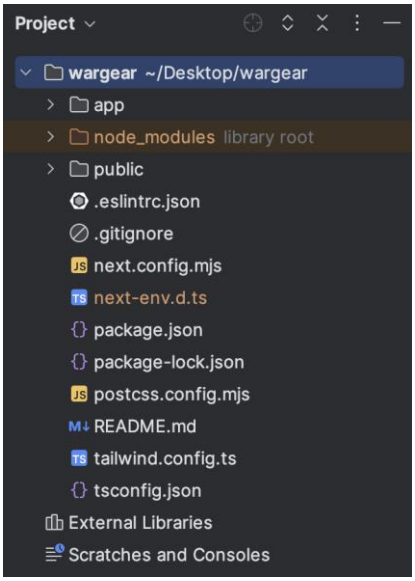


Рисунок 4.4 — Початкова структура проекту

Директорія “app” є початковою точкою входу нашого веб-сайту, всередині неї в майбутньому будуть знаходитись директорії, які будуть відповідати за сторінки веб-порталу. Директорія “node_modules” автоматично створюється в проекті JavaScript. Вона містить всі залежності та модулі, які потрібні для роботи проекту. Також при ініціалізації проекту за допомогою команди “create-next-app” автоматично ініціалізується Git-репозиторій і створюється файл “.gitignore”, що відповідає за ігнорування файлів, які не потрібно зберігати в git, наприклад конфігураційні файли середовища розробки.

Крім того, варто звернути увагу на конфігураційні файли проекту. Файл “next.config.mjs” містить налаштування для Next.js, які можуть включати налаштування збірки, оптимізацію зображень та інші параметри. Файл “tailwind.config.ts” дозволяє налаштовувати Tailwind CSS відповідно до потреб проекту, додаючи кастомні кольори, шрифти та інші стилі. Файли конфігурації TypeScript “tsconfig.json” та ESLint “.eslintrc.json” забезпечують типізацію та статичний аналіз коду, допомагаючи підтримувати високу якість коду та уникати помилок.

Отже, проект повністю готовий та має всі потрібні допоміжні інструменти для подальшої роботи та створення програмного продукту.

4.4 Структура бази даних

В цьому підрозділі ми розглянемо структуру бази даних веб-порталу підбору військової амуніції. Оскільки для зберігання даних використовується нереляційна NoSQL база даних MongoDB, вона містить складні структури такі як, масиви, вкладені документи, на відміну від звичайних реляційних баз даних, які містять прості структури і моделі описуються таблицями [14]. Для наочності і розуміння, які моделі, поля та зв’язки між сутностями містить база даних, зобразимо її схематично та зробимо певну декомпозицію, щоб спростити складні структури MongoDB. Структура БД зображена на рисунку 4.5.

Рисунок 4.5 — Структура бази даних у вигляді таблиць

Для зручності розробки та забезпечення цілісності даних, структура даних була продумана таким чином, щоб кожна колекція мала чітко визначені поля та зв’язки з іншими колекціями.

База даних містить наступні колекції:

- Users, яка зберігає інформацію про користувачів веб-порталу;
- Products, зберігає дані про товари розміщені на порталі;
- Orders, зберігає інформацію про замовлення оформлені користувачами.

Для колекції Orders була виконана декомпозиція в цій схемі на Orders та OrderItem, так як колекція Orders містить інформацію про товари та додаткові опції замовлення, що знаходяться в полі items у вигляді масиву. Також в колекціях є поля які мають значення типу об’єкт, така структура була спрощена в схемі та описана окремими полями для кожної властивості об’єкту, наприклад поля в колекції Orders з приставкою shippingAddress.

В структурі БД існують такі зв’язки між колекціями, а саме між Users та Orders зв’язок “один до багатьох”, це означає що один користувач може мати декілька замовлень, поле “userId” в колекції Orders посилається на “_id” в колекції Users. Між Products та OrderItems існує зв’язок “один до багатьох”, тому що один товар може бути в багатьох елементах замовлення. Між структурами Orders та OrderItem є зв’язок “один до багатьох”, завдяки тому що одне замовлення може містити багато елементів замовлення.

4.5 Розробка back-end частини

Зрозумівши, який функціонал буде в майбутньому веб-порталі, можна розробляти back-end частину проекту. В цьому підрозділі розглянемо деталі та особливості створення бази даних MongoDB та під’єднання до неї. Також описано структуру API роутингу в Next.js та основні ендпоінти системи.

Для початку потрібно створити кластер на MongoDB, щоб зробити базу даних для зберігання інформації з сервісу. Після створення кластеру отримуємо MongoDB URI для підключення до бази даних в проекті. Цей ідентифікатор потрібно записати в файл “.env”, в якому зберігаються змінні середовища. Взаємодія з базою даних MongoDB відбувається через інструмент mongoose [15]. Щоб проводити будь-які операції з базою спочатку потрібно виконати під’єднання, для цього створена відповідна функція (рисунок 4.6).

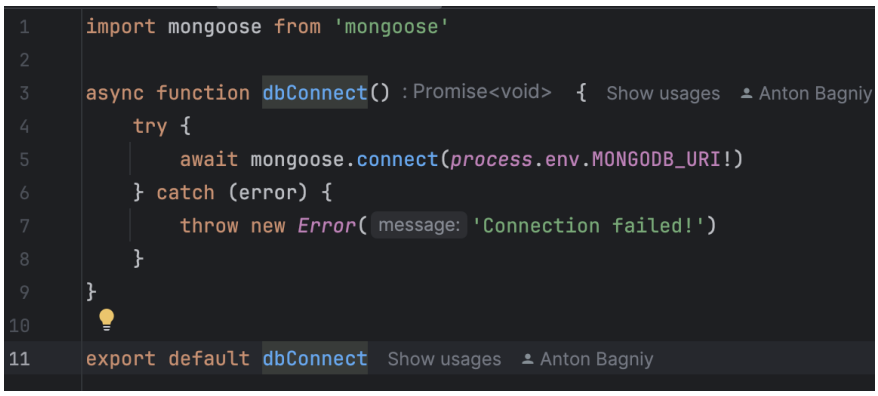
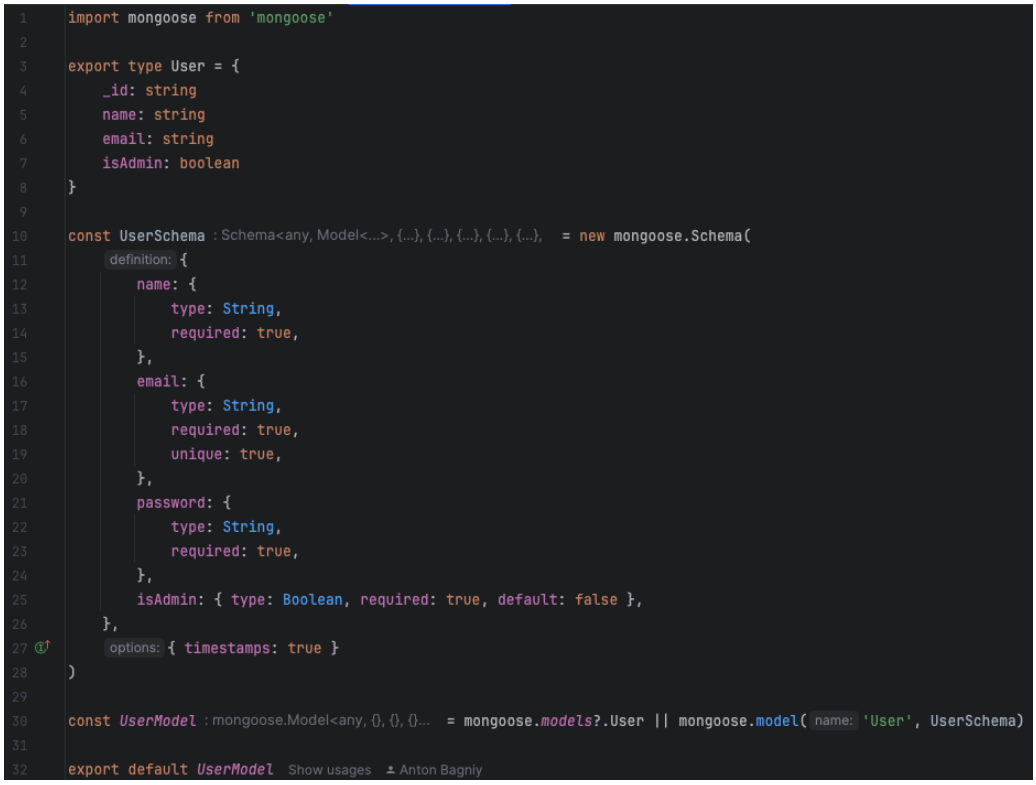


Рисунок 4.6 — Функція під’єднання до БД

Тепер перед будь-якою взаємодією з базою даних потрібно викликати функцію для з’єднання. У випадку якщо під час з’єднання виникне якась проблема, з’явиться помилка, що підключення не вдалось.

Наступним кроком створимо моделі. Моделі представляють собою структури даних, які визначають, як зберігати інформацію в базі даних. Вони дозволяють визначити, які поля повинен мати той чи інший об’єкт, якого типу кожне поле, чи обов’язкове воно та які правила валідації потрібно застосовувати. Також моделі дають можливість визначити взаємозв’язки між сутностями. Взагалі вони забезпечують чітку структуру даних, що полегшує роботу з ними. Для створення моделей ми скористаємось інструментами, які надає MongoDB. Кожна модель буде представлена у вигляді схеми, яка визначає структуру документа, що зберігається в колекції.

В проєкті створені три моделі: користувач, товар та замовлення. Оскільки повинно бути реалізовано авторизацію користувачів, то потрібно щоб була модель, яка буде визначати основні атрибути користувачів, такі як ім’я, електронна пошта, пароль та роль, а саме чи є користувач адміністратором. Модель користувача визначає, які поля повинні бути в об’єкті користувача та правила валідації для кожного поля. Роль вказується типом boolean, якщо користувач є адміном то значення для поля “isAdmin” буде true. Пароль зберігається в БД у зашифрованому вигляді завдяки бібліотеки bcrypt, її функція зашифровує пароль перед записом в базу даних. Реалізація схеми та моделі зображена на рисунку 4.7.



```
import mongoose from 'mongoose'

export type User = {
    _id: string
    name: string
    email: string
    isAdmin: boolean
}

const UserSchema : Schema<any, Model<...>, {...}, {...}, {...}, {...}, {...},   = new mongoose.Schema(
     definition: {
        name: {
            type: String,
            required: true,
        },
        email: {
            type: String,
            required: true,
            unique: true,
        },
        password: {
            type: String,
            required: true,
        },
        isAdmin: { type: Boolean, required: true, default: false },
    },
     options: { timestamps: true }
)

const UserModel : mongoose.Model<any, {}, {}, {}... = mongoose.models?.User || mongoose.model( name: 'User', UserSchema)

export default UserModel  Show usages  Anton Bagniy
```

Рисунок 4.7 — Модель користувача

Щоб створити модель спочатку за допомогою бібліотеки mongoose створюється схема з відповідними полями, визначенням типу та обов’язковістю. Кожне поле яке існує в об’єкті схеми є атрибутом моделі. Потім на основі схеми створюється модель. Якщо така модель вже існує то вона повертається, якщо ж ні, то створюється нова.

Для опису сутності товару створена модель “Product”, рисунок 4.8.

```
const productSchema : Schema<any, Model<...>, {...}, {...}, {...}, {...}, {...},   = new mongoose.Schema(
     definition: {
        name: { type: String, required: true },
        slug: { type: String, required: true, unique: true },
        category: { type: String, required: true },
        image: { type: String, required: true },
        price: { type: Number, required: true },
        brand: { type: String, required: true },
        sizes: { type: String, required: true },
        colors: { type: String, required: true },
        rating: { type: Number, required: true, default: 0 },
        numReviews: { type: Number, required: true, default: 0 },
        countInStock: { type: Number, required: true, default: 0 },
        description: { type: String, required: true },
        isFeatured: { type: Boolean, default: false },
        banner: String,
    },
     options: {
        timestamps: true,
    }
)
```

Рисунок 4.8 — Модель товару

В цій схемі знаходяться всі поля які повинні бути в об’єкті товару. В деяких полях вказано значення за замовчуванням, це корисно якщо певний атрибут обов’язково має містити якесь значення. Другим параметром який отримує schema є певні налаштування. Наприклад в даному випадку ми встановлюємо timestamps, це означає що для кожного об’єкту будуть автоматично додаватись поля з датою створення та оновлення.

Щоб описати сутність замовлення була створена відповідна схема та на основі неї модель (рисунок 4.9).

```
const orderSchema : Schema<any, Model<...>, {...}, {...}, {...}, {...}, {...}, {...}> = new mongoose.Schema(
    definition: {
        user: {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'User',
            required: true,
        },
        items: [
            {
                product: {type: mongoose.Schema.Types.ObjectId...},
                name: { type: String, required: true },
                slug: { type: String, required: true },
                color: { type: String, required: true },
                size: { type: String, required: true },
                qty: { type: Number, required: true },
                image: { type: String, required: true },
                price: { type: Number, required: true },
            },
        ],
        shippingAddress: {
            fullName: { type: String, required: true },
            address: { type: String, required: true },
            city: { type: String, required: true },
        },
        paymentMethod: { type: String, required: true },
        paymentResult: { id: String, status: String, email_address: String },
        itemsPrice: { type: Number, required: true },
        shippingPrice: { type: Number, required: true },
        taxPrice: { type: Number, required: true },
```

Рисунок 4.9 — Модель замовлення

Модель містить всі типові поля, які необхідні для замовлення в інтернет-магазині, а саме прив’язка до користувача який зробив замовлення, масив товарів, які були додані в замовлення, адреса доставки, метод оплати та інші. За допомогою ключового слова “ref” і виконується прив’язка до інших моделей.

В результаті на рисунку 4.10 можемо побачити створені колекції даних на основі моделей в програмному забезпеченні “MongoDB Compass”.

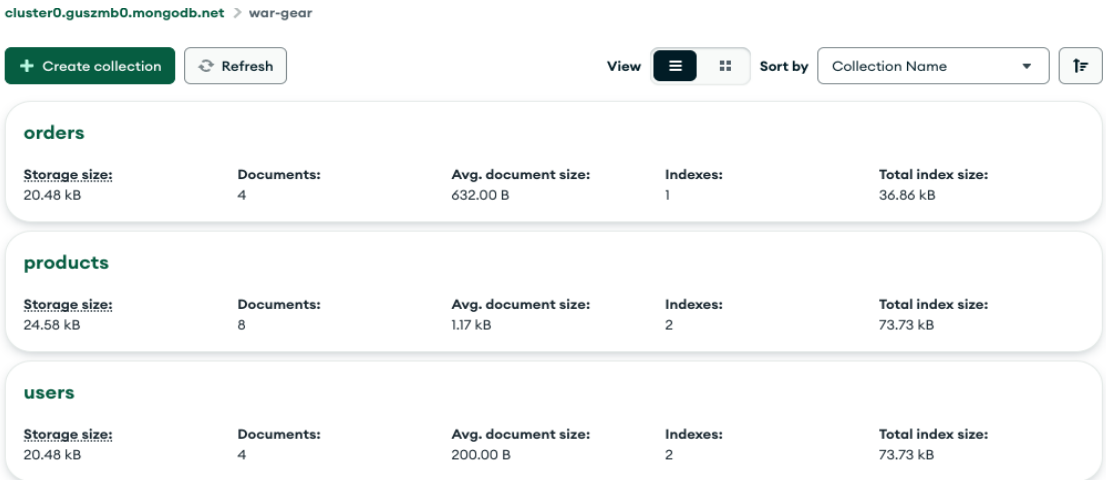


Рисунок 4.10 — Створені колекції в MongoDB

Для того щоб взаємодіяти з моделями та даними, які знаходяться в базі даних використовується API роутинг [16]. Він в Next.js працює через файлову структуру в директорії “app/api”. Кожен файл у цій директорії відповідає за обробку запитів до певного маршруту. Назва директорії визначає endpoint маршруту. Розглянемо створену схему роутів (рисунок 4.11).

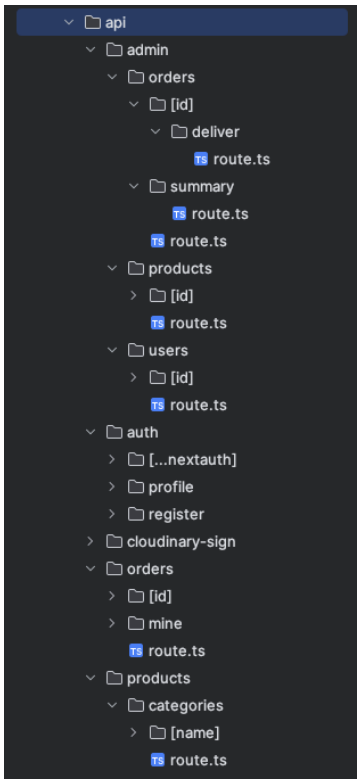


Рисунок 4.11 — Структура директорії api

Директорія “admin” відповідає за роути, які будуть використовуватись в адмін панелі, для взаємодії з замовленнями, товарами та користувачами. Наприклад для переглядання всіх замовлень, інформації по одному замовленню, статистичних даних та зміни статусу доставки, додавання товару, зміни даних товару, переглядання товарів. Для того щоб можна було отримати дані, наприклад по одному конкретному замовленню, в Next.js існує можливість створення динамічного маршруту. Це реалізується за рахунок назви директорії, вона має бути в квадратних дужках. Тобто в даному випадку “[id]” – динамічний маршрут для отримання інформації про замовлення за його ідентифікатором.

Директорія “auth” використовується для взаємодії з даними користувача, а саме реєстрація та зміна даних про користувача.

Для завантаження зображень товарів, була виконана інтеграція сервісу Cloudinary. Працює це таким чином що ми відправляємо зображення на сервер сервісу а тоді вже посилання на зображення на сервері записуємо в базу даних до відповідного товару.

Директорія “orders” створена для групування маршрутів, які відповідають за роботу з замовленнями користувача, тобто для отримання всіх замовлень, отримання даних про конкретне замовлення по ідентифікатору та створення замовлення.

Директорія “products” відповідно групує маршрути, які використовуються для роботи з товарами.

Для прикладу розглянемо ендпоінт на отримання даних про конкретне замовлення (рисунок 4.12).

```
export const GET = auth(async (...request: any) : Promise<Response> => {  no usages  ± Anton Bagniy
    const [req, { params }] = request
    if (!req.auth) {
        return Response.json(
            data: { message: 'unauthorized' },
            init: {
                status: 401,
            }
        )
    }
    await dbConnect()
    const order = await OrderModel.findById(params.id)
    return Response.json(order)
}) as any
```

Рисунок 4.12 — Запит на отримання конкретного замовлення

Спочатку оголошуємо асинхронну функцію “GET”, це явно вказує тип запиту. “auth” це middleware для перевірки автентифікації користувача перед виконанням основної логіки. Далі отримуємо аргументи функції, як масив. Перший аргумент в масиві це запит, а другий – об‘єкт з параметрами. Якщо користувач не авторизований то запит поверне помилку “401” з відповідним текстом. Наступним кроком йде підключення до бази даних за допомогою функції dbConnect, яку створили раніше. Виконуємо пошук замовлення в базі даних за ідентифікатором та повертаємо знайдене замовлення.

Ще розглянемо отримання всіх товарів, з реалізованим функціоналом пошуку та фільтрації (рисунок 4.13).

```
export const GET = async (req: NextRequest) : Promise<Response>  => {  no usages  Anton Bagniy
    await dbConnect()
    const url : URL  = new URL(req.url)

    const search : string | null  = url.searchParams.get("search")
    let filter: Filter = { name: { $regex: `${search}`, $options: 'i' } }

    const lte : string | null  = url.searchParams.get("lte")
    const gte : string | null  = url.searchParams.get("gte")
    const color : string  = `${url.searchParams.get("color")}`
    const size : string  = `${url.searchParams.get("size")}`

    if (size !== null && size !== '') {
        filter.sizes = { $regex: `${size}`, $options: 'i' }
    }

    if (color !== null && color !== '') {
        filter.colors = { $regex: `#${color}`, $options: 'i' }
    }

    if (lte !== null && !isNaN(parseFloat(lte))) {
        filter.price = { ...filter.price, $lte: parseFloat(lte) }
    }

    if (gte !== null && !isNaN(parseFloat(gte))) {
        filter.price = { ...filter.price, $gte: parseFloat(gte) }
    }

    const products : (FlattenMaps<any> & Required<{...  = await ProductModel.find(filter).lean();
    return Response.json(products)
}
```

Рисунок 4.13 — Запит на отримання всіх товарів з фільтрацією та пошуком

В даному випадку, виконуємо всі такі ж дії як в попередньому запиті, але особливість полягає в тому що при надсиланні запиту в url ендпоінта можна додавати searchParams. Завдяки цьому ми можемо отримати значення з searchParams по ключу та зробити фільтрацію. Таким чином ключ search відповідає за пошук по назві товару, lte та gte за фільтрацію по ціні, а size та color за розмір та

колір. Повертаємо товари, які підходять під параметри, якщо параметри порожні то повертаються всі товари.

Також розглянемо ендпоінт для видалення конкретного товару по id. Реалізація ендпоінта на видалення товару зображена на рисунку 4.14.

```
await dbConnect()
const product = await ProductModel.findById(params.id)
if (product) {
    await product.deleteOne()
    return Response.json( data: {message: 'Product deleted successfully'})
} else {
    return Response.json(
        data: {message: 'Product not found'},
        init: {
            status: 404,
        }
    )
}
```

Рисунок 4.14 — Запит на отримання всіх товарів з фільтрацією та пошуком

Видаляти товари в системі може тільки адміністратор, тому цей запит теж перевіряє чи авторизований користувач під роллю адміністратора, в інакшому випадку front-end отримає помилку 401 – неавторизовано. Наступним кроком асинхронно під’єднуємось до БД і в колекції товарів шукаємо потрібний товар по id за допомогою асинхронної функції “findById”, яку надає нам mongoose. Якщо такий товар був знайдений, то виконуємо знову асинхронну функцію “deleteOne”, яка видаляє один елемент з колекції. У випадку успішності виконання функцій, запит повертає відповідне повідомлення та статус-код 200, що означає успішне виконання запиту. Якщо такого товару знайдено не було, то клієнтська частина отримує статус-код 404 та повідомлення, що товар не знайдений.

4.6 Розробка front-end частини

Після розробки back-end частини приступаємо до створення front-end, а саме сторінок та функціональних компонентів з яких будуть складатись сторінки і виконаємо підключення API.

Next.js також має особливості при побудові роутингу і для front-end, він використовує файлову структуру для автоматичного створення маршрутів. Кожна директорія у папці “app” відповідає окремій сторінці та маршруту на сайті. Це робить роутинг дуже інтуїтивним та зручним. Також фреймворк дозволяє створювати макети (layouts), які можна використовувати для організацій спільних частин інтерфейсу, наприклад header, footer, nav, sidebar та інші.

Розглянемо структуру створеної front-end частини (рисунок 4.15).

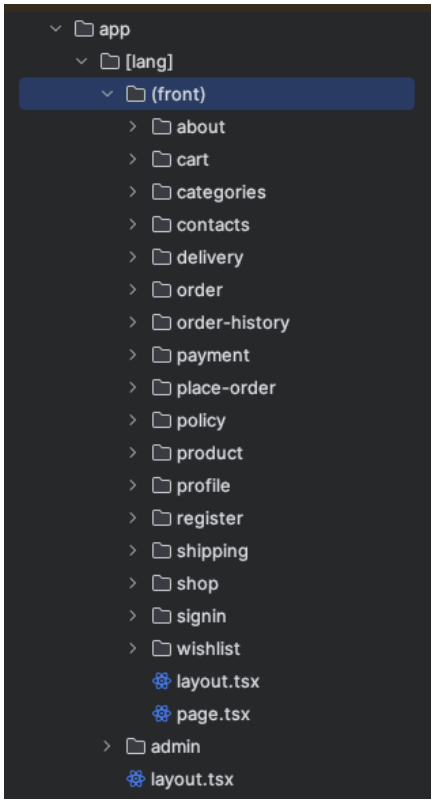


Рисунок 4.15 — Структура front-end

Всі директорії сторінок знаходяться в динамічній директорії [lang]. Все через те що в проекті реалізована локалізація за допомогою Next.js, а саме інструмента i18n. Це означає що в маршруті до будь-якої сторінки буде елемент шляху який показує на якій мові ми зараз знаходимось. Далі по ієрархії в директорії [lang] знаходяться дві директорії: front та admin, які використовуються для групування сторінок. Дужки в назві директорії означають що в маршруті ця назва відображатись не буде. Це було зроблено щоб для адмін панелі layout відрізнявся від інших звичайних сторінок.

Інтерфейс звичайного користувача має наступні сторінки:

– /about: сторінка з інформацією про магазин;
– /cart: сторінка кошика покупок;
– /categories: сторінка з категоріями товарів;
– /contacts: сторінка контактів;
– /delivery: сторінка з інформацією про доставку;
– /order: сторінка з деталями замовлення;
– /order-history: сторінка історії замовлень користувача;
– /payment: сторінка оплати;
– /place-order: сторінка оформлення замовлення;
– /policy: сторінка з політиками компанії;
– /product: сторінка товару;
– /profile: сторінка профілю користувача;
– /register: сторінка реєстрації користувача;
– /shipping: сторінка для вибору способу доставки;
– /shop: сторінка всіх товарів;
– /signin: сторінка входу в систему;
– /wishlist: сторінка з обраними товарами користувача.

Інтерфейс адміністратора має такі сторінки:

– /dashboard: сторінка зведених даних та статистики;
– /orders: сторінка замовлень;
– /products: сторінка товарів, розміщених на порталі;
– /users: сторінка зі всіма користувачами.

Директорії categories, order, product, products та users містять в собі динамічні директорії, для отримання динамічних даних по певному ідентифікатору.

Розглянемо створення функціоналу кошика. Для початку створимо сторінку “/cart”, рисунок 4.16. Опишемо параметри які приймає функція сторінки, а саме вкажемо що в параметрах є params, це об’єкт даних які знаходяться в динамічних роутах. В даному випадку сторінка “cart” знаходиться в динамічному роуті “lang”, тому ми можемо з params дістати значення “lang”.

```
export const metadata : Metadata  = {  no usages  Anton Bagniy
    title: 'Кошик',
}

const CartPage = async ({params}: { params: {lang: Locale}}) : Promise<Element>  => {
    const dict : {sidebar: {shop: string, categ...  = await getDictionary(params.lang);

    return <CartDetails lang={params.lang} dict={dict}/>
}

export default CartPage  no usages  Anton Bagniy
```

Рисунок 4.16 — Сторінка кошика

Для кращої SEO оптимізації було описано метатег для цієї сторінки. У Next.js є можливість створювати як серверні, так і клієнтські компоненти. Серверні компоненти рендеряться на сервері при кожному запиті і дозволять швидше відображати контент для користувачів та забезпечують кращу SEO-оптимізацію, оскільки пошукові системи отримують повністю відрендерену сторінку. Всі компоненти в Next за замовчуванням є серверними, щоб вказати що це клієнтський компонент то потрібно написати ключові слова “use client”. В моєму випадку оскільки це сторінка, то краще її залишити серверною, а всю логіку з хуками перенести в окремий компонент “CartDetails”. На кожній сторінці з динамічного роуту потрібно діставати з params мову, для визначення яка мова вибрана в даний момент на сайті. Для отримання словника з текстом вибраної мови була створена функція getDictionary.

```
export type Dictionary = typeof tmp;

const dictionaries : {en: () => Promise<{sidebar: {shop:..., uk: () => Promise<{sidebar: {shop:...}  = {
  en: () => import('@/dictionaries/en.json').then((module) => module.default),
  uk: () => import('@/dictionaries/uk.json').then((module) => module.default),
};

export const getDictionary = async (locale: Locale) : Promise<{sidebar: {shop: string, categ...>  => dictionaries[locale]();
```

Рисунок 4.17 — Функція getDictionary

По переданій в функцію локалі вона повертає словник зі всіма текстами та словами, які внесли раніше.

Для зберігання даних про товари, які були додані в кошик та зміни кількості товару в ньому використовується store з бібліотеки Zustand. На рисунку 4.18

зображена реалізація кнопок для збільшення та зменшення конкретного товару в кошику.

```
<div
    className="inline-flex items-center px-4 font-semibold text-gray-500 border border-gr
    <Button className="mr-1" size="icon" variant="ghost"
            onClick={() => decrease(item)}> - </Button>
    <span>{item.qty}</span>
    <Button className="ml-1" size="icon" variant="ghost"
            onClick={() => increase(item)}> + </Button>
</div>
```

Рисунок 4.18 — Кнопки для збільшення та зменшення кількості товарів

В компоненті кошика застосовуються функції які визначені в сторі для зміни кількості товару. Для прикладу розглянемо функцію для збільшення кількості товару, вона продемонстрована на рисунку 4.19.

```
increase: (item: OrderItem) : void  => {
    const exist : OrderItem | undefined  = items.find(
        (x : OrderItem ) =>
            x.slug === item.slug && x.color === item.color && x.size === item.size
    )
    let updatedCartItems : OrderItem[]  = exist
        ? items.map((x : OrderItem ) : {...}  =>
            x.slug === item.slug && x.color === item.color && x.size === item.size
                ? { ...exist, qty: exist.qty + 1 }
                : x
        )
        : [...items, { ...item, qty: 1, color: item.color, size: item.size }]
    const { itemsPrice : number , shippingPrice : number , taxPrice : number , totalPrice : number  } =
        calcPrice(updatedCartItems)
    cartStore.setState( partial: {
        items: updatedCartItems,
        itemsPrice,
        shippingPrice,
        taxPrice,
        totalPrice,
    })
},
```

Рисунок 4.19 — Функція для збільшення кількості товарів

Для початку знаходимо існуючий товар методом “find” з тим самим “slug”, “color”, “size”. Якщо товар існує, то створюється новий масив “updatedCartItems”, де для існуючого товару збільшується кількість на одиницю. Якщо товар не існує, додається новий товар з початковою кількістю 1 до масиву “items”. Далі викликається функція “calcPrice”, яка обчислює ціну, а саме вартість товарів, доставки, податок та загальну суму відповідно до нових даних. В кінці метод

“cartStore.setState” оновлює стан стору з новими даними. Аналогічно працює функція “decrease”.

Для стилізації компонентів інтерфейсу використовується Tailwind. Це CSS бібліотека, яка дозволяє швидко та ефективно створювати стильні компоненти користувацького інтерфейсу.

Щоб отримати текст мовою, яка вибрана на сайті, з словника dict переходимо до поля в якому визначений потрібний текст, як показано на рисунку 4.20. Таким чином весь текст який знаходиться на сайті має знаходитись в файлах-словниках.

```
<h2 className="mb-8 text-4xl font-bold">{dict.cartDetails.yourCart}</h2>
```

Рисунок 4.20 — Приклад отримання тексту з словника

Розглянемо компонент Shop, в якому описаний весь функціонал сторінки “/shop”. Спочатку потрібно отримати товари з бази даних використовуючи ендпоінт “/api/products”. Отримання товарів, відправляючи запит на відповідний endpoint, зображено на рисунку 4.21.

```
useEffect( effect: () : void => {
    const fetchData = async () : Promise<void> => { Show usages  Anton Bagniy *
        try {
            const response : Response = await fetch( input: `/api/products?search=${search}&gte=${gte}&lte=${lte}&color=${selectedColor}&size=${selectedSize}`);
            if (!response.ok) {
                throw new Error( message: 'Network response was not ok');
            }
            const data = await response.json();
            setProducts(data);
        } catch (error) {
            console.error('Error fetching data:', error);
        }
    };

    fetchData();
}, deps: [search, gte, lte, selectedColor, selectedSize]);
```

Рисунок 4.21 — Отримання товарів

Для виконання цієї операції використовується хук useEffect, для того щоб запит відправлявся при першому рендері компонента. Хук приймає два аргументи, перший це callback функція, яка буде виконуватись при певній умові, а другий це масив залежностей. Це означає що callback функція буде виконуватись при першому рендері та при зміні будь-якого елемента в масиві залежностей [17]. В якості залежностей було записано стани пошуку та параметрів фільтру, зроблено

це з метою щоб коли користувач вибирав якісь фільтри то запит на отримання товарів відправлявся знову. При запиті в ендпоінт підставляються значення фільтрів та пошуку в searchParams.

Після отримання товарів з бази даних, виконуємо їхній рендер за допомогою методу масивів “map” [18] (рисунок 4.22).

```
{products.map((product: Product, productIndex: number) => (
    <div
        key={productIndex}
        className="w-[338px] self-center justify-self-center bg-primary-foreground shadow-md rounded-xl duration-500 hover:scale-105 hover:shadow-xl">
        <Link href={`/${lang}/product/${product.slug}`}>
            <Image
                width={338}
                height={320}
                src={product.image}
                alt="Product" className="h-80 w-[338px] object-cover rounded-t-xl"/>
            <div className="px-4 py-3 w-72">
                <span
                    className="text-gray-400 mr-3 uppercase text-xs">{product.brand || 'WarGear'}</span>
                <p className="text-lg font-bold text-primary     block capitalize">{product.name}</p>
                <div className="flex items-center">
                    <p className="text-lg font-semibold text-primary cursor-auto my-3">{product.price}{dict.uah}</p>
                    <del>
                        <p className="text-sm text-primary/80 cursor-auto ml-2">{(product.price * 1.1).toFixed( fractionDigits: 1)}{dict.uah}</p>
                    </del>
                </div>
            </div>
        </Link>
    </div>
))}
```

Рисунок 4.22 — Рендеринг товарів

Цей метод проходиться по масиву товарів та повертає кожен елемент масиву. Потім дані конкретного елементу масиву підставляються у верстку.

Отже в цьому розділі було описано принципи та особливості програмної реалізації основних елементів та функцій веб-порталу, від підключення до БД та створення моделей до створення інтерфейсу та отримання даних з бази даних.

5 РОБОТА КОРИСТУВАЧА З ВЕБ-ПОРТАЛОМ

В цьому розділі описана робота користувача з веб-порталом, як виглядає створений інтерфейс та результати роботи. Окрім цього, буде розглянуто інтерфейс основних функціональних можливостей порталу та основні етапи взаємодії користувача з системою.

Перейшовши на сайт веб-порталу, користувача зустрічає головна сторінка, зображена на рисунку 5.1.

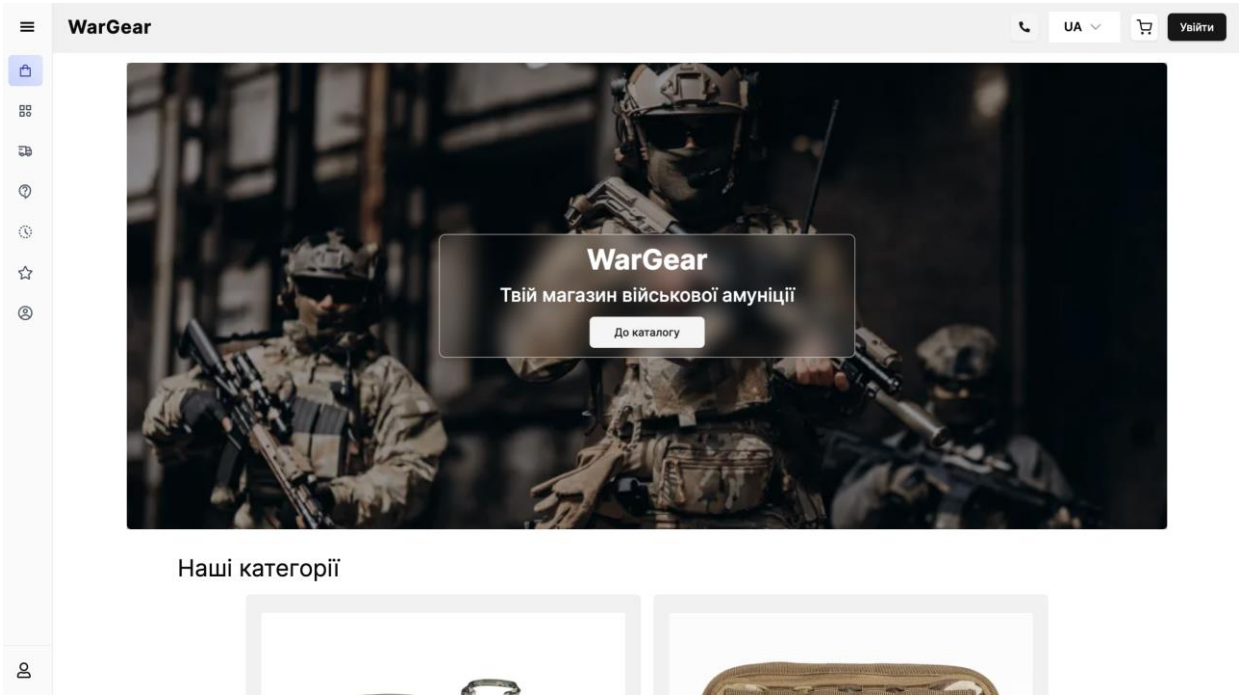


Рисунок 5.1 — Головна сторінка WarGear

Інтерфейс складається з хедера в якому розміщено меню з телефоном та поштою для зв’язку, перемикач мови, кнопка для переходу в кошик та кнопка для авторизації.

Бічне меню має здатність розширюватись та згортатись, що заощаджує місце і не забирає зайву увагу (рисунок 5.2). Також такий підхід є оптимальним для веб-порталів адаптованих для різних пристроїв, таким чином зручність користування буде забезпечено і для невеликих пристроїв.

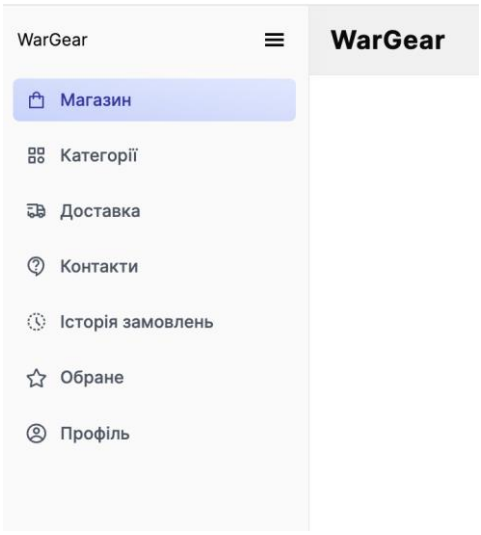


Рисунок 5.2 — Розширене бічне меню

В меню знаходяться основні сторінки які потрібні для звичайного користувача. Далі користувач зразу бачить категорії товарів, які є в магазині і може перейти до конкретної категорії за потреби. Також є кнопка “До каталогу”, яка направляє на сторінку зі всім асортиментом товарів.

Якщо натиснути на кнопку “Увійти” користувач потрапляє на сторінку входу в обліковий запис (рисунок 5.3).

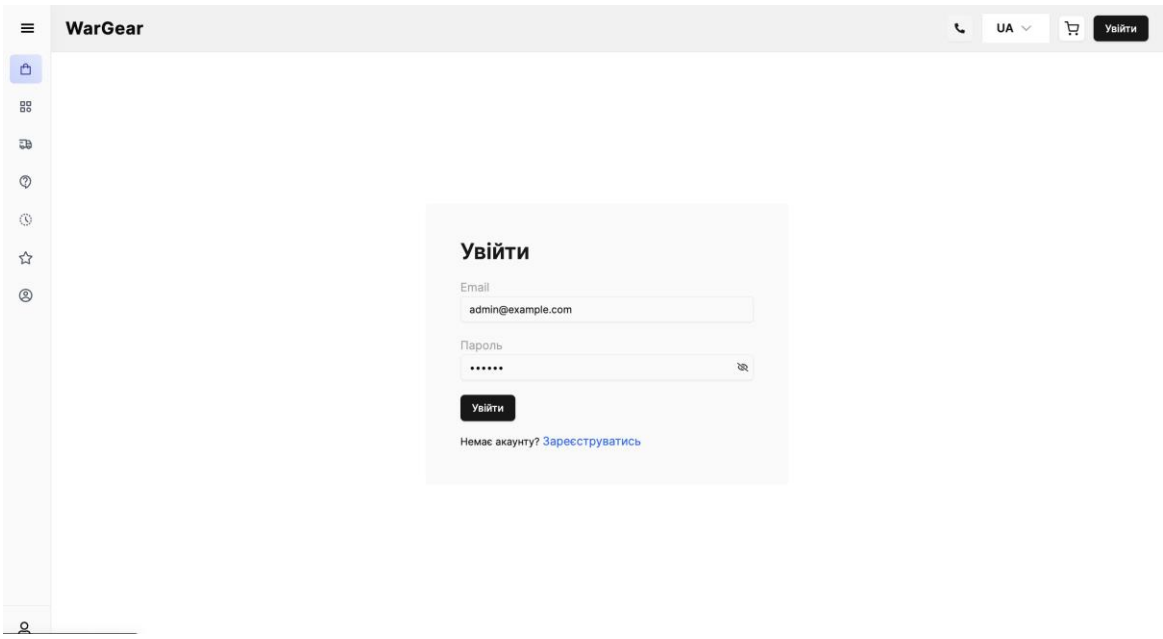


Рисунок 5.3 — Сторінка входу

Йому пропонується ввести Email та пароль. Якщо користувач ввів неправильні дані, то виведеться відповідна помилка з зрозумілим текстом, що саме користувач зробив не так.

Якщо ж це новий користувач то можна перейти на сторінку рестрації (рисунок 5.4).

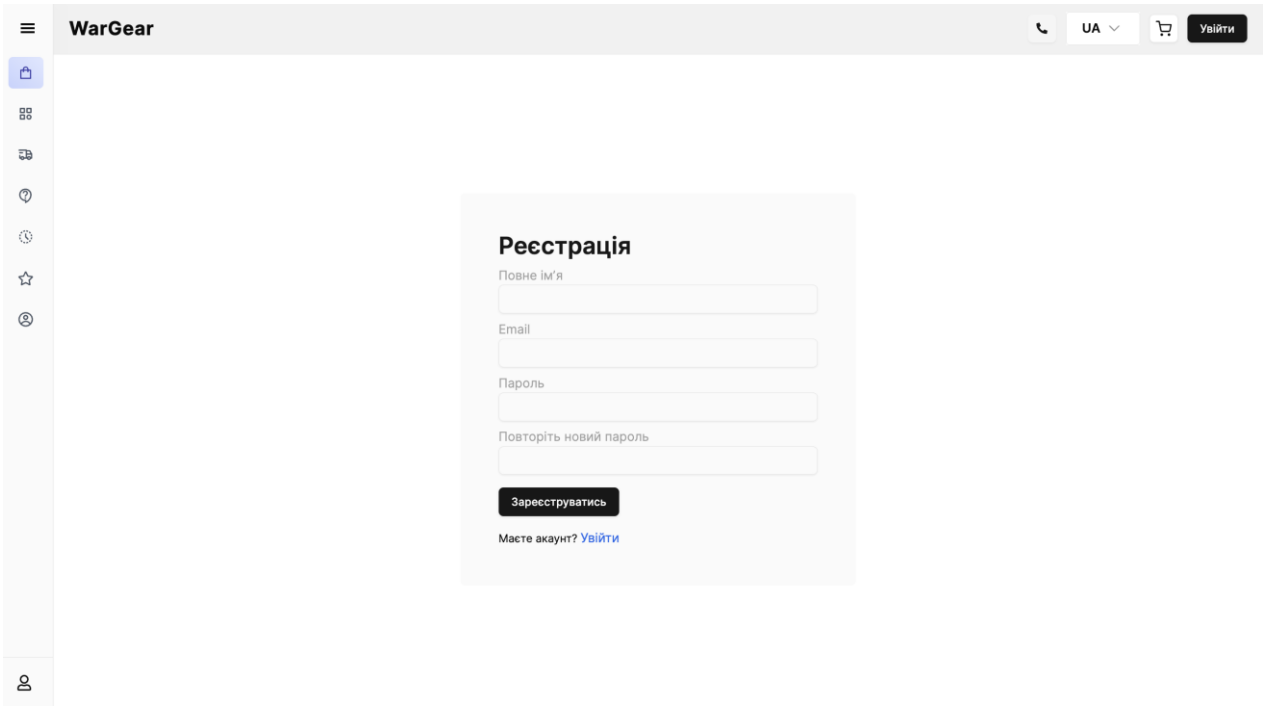


Рисунок 5.4 — Сторінка реєстрації

Після введення всіх необхідних даних та підтвердження реєстрації, натисканням на кнопку “Зареєструватись”, користувач з’являється в системі і тоді вже відкривається повний доступ, а саме можна робити замовлення та переглядати свої замовлення.

Якщо перейти до сторінки всього асортименту користувач побачить всі товари, які є в магазині (рисунок 5.5). Це дуже зручно, так як якщо користувач не шукає якийсь конкретний товар, він може проглянути весь асортимент і вже тоді щось підбирати.

Кожен товар супроводжується зображенням, назвою бренду, коротким описом товару та ціною. Такий набір інформації не перевантажує користувача і є основним.

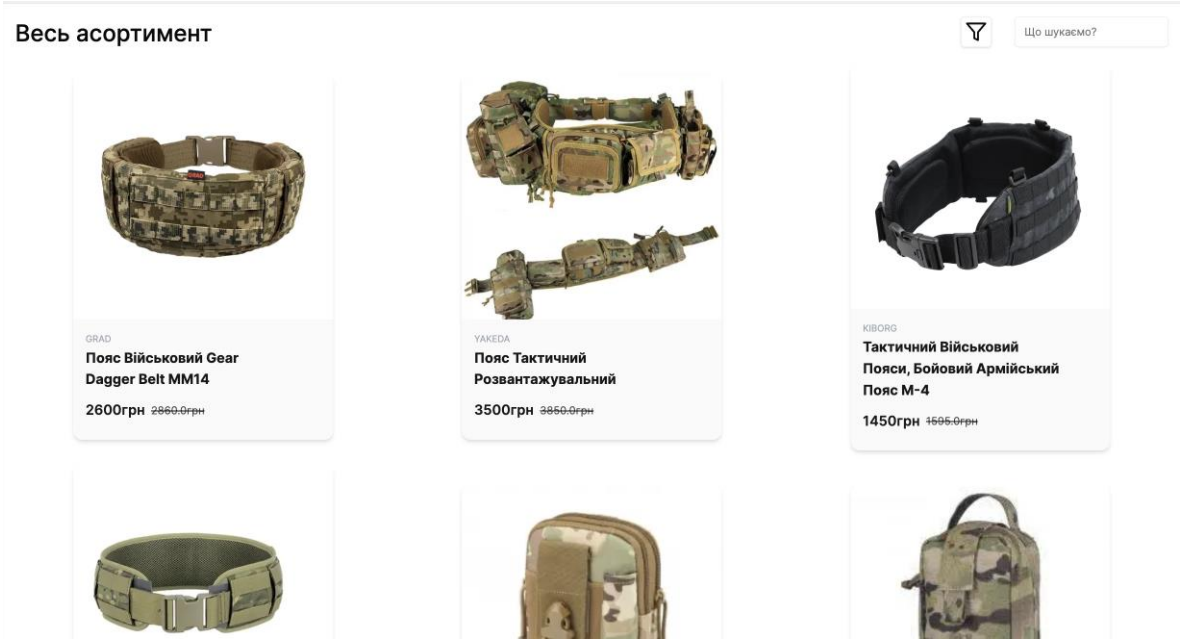


Рисунок 5.5 — Сторінка “Магазин”

В правій частині є пошук, куди користувач може ввести назву товару, бренд або інше пов’язане з назвою товару. Натиснувши на значок фільтру, з’являться поля куди можна ввести параметри за якими потрібно відфільтрувати товари (рисунок 5.6).

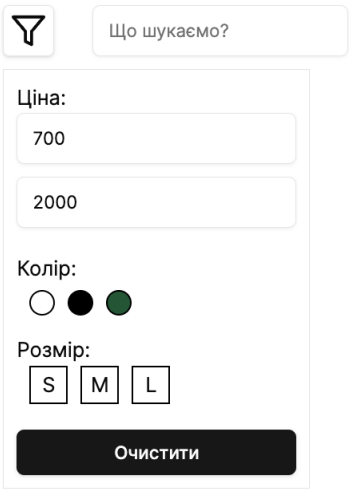


Рисунок 5.6 — Відкритий фільтр з пошуком

Також є кнопка “Очистити”, щоб прибрати параметри та показати всі товари.

Для того щоб побачити детальну інформацію про конкретний товар, користувачу потрібно натиснути на відповідну картку товару. Після переходу він

побачить зображення товару, назву, ціну, наявність, колір, розмір та опис товару (рисунок 5.7).

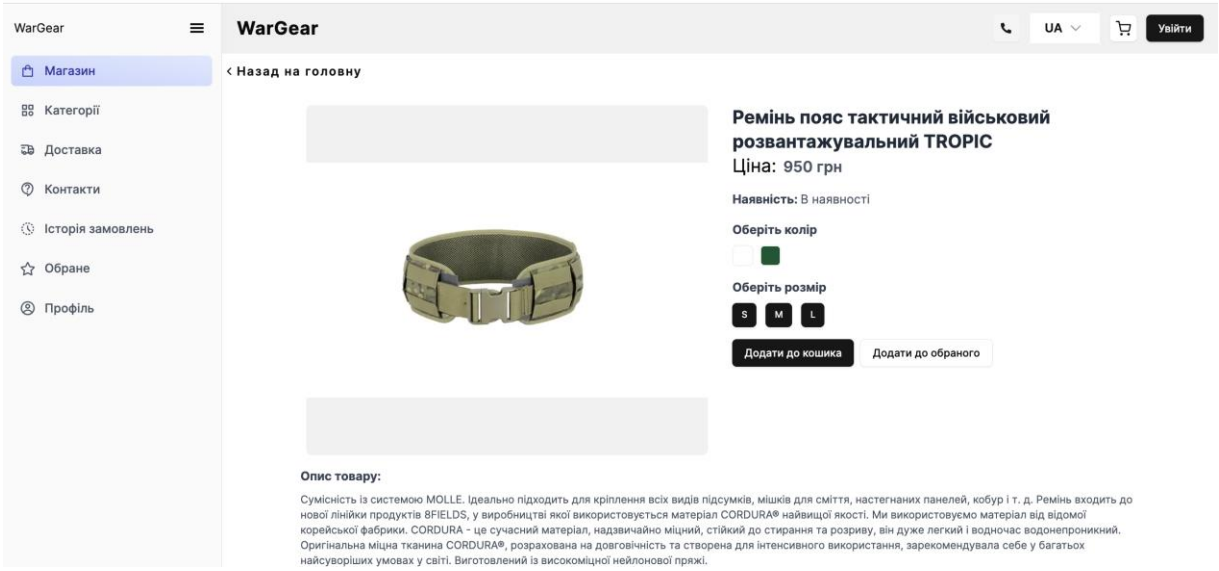


Рисунок 5.7 — Сторінка товару

Після вибору кольору та розміру товару є можливість додати його до кошика або до обраного, якщо користувач ще не хоче робити замовлення, а тільки придивляється. Якщо він додав до обраного то в бічному меню з’являється позначка скільки товарів знаходиться в обраному (рисунок 5.8).

Рисунок 5.8 — Позначка обраних товарів

Перейшовши до обраного бачимо доданий товар і також є можливість очистити весь список натиснувши на відповідну кнопку (рисунок 5.9). Якщо натиснути щоб додати такий самий товар до обраного, то він просто залишиться в списку. Додаючи інші товари, вони формують сітку схожу як на сторінці “Магазин”. Якщо користувач закриє сторінку і потім знову відкриє, то список обраних збережеться, така поведінка відбувається завдяки стору з бібліотеки Zustand.

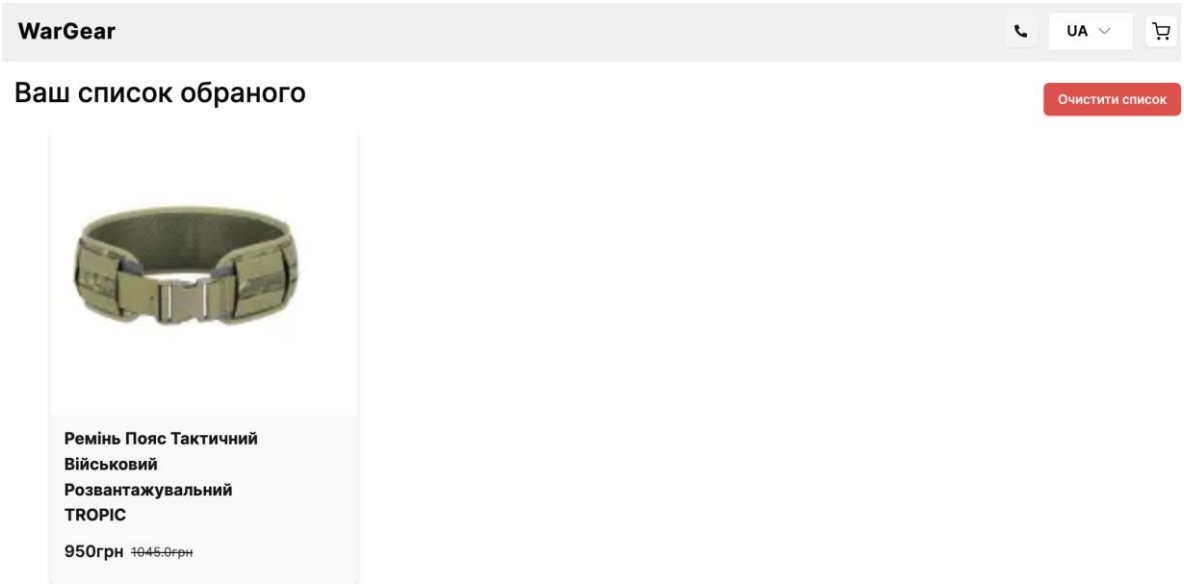


Рисунок 5.9 — Позначка обраних товарів

При додаванні товару до кошика теж з’являється позначка про кількість доданих товарів (рисунок 5.10).

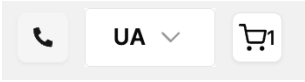


Рисунок 5.10 — Позначка обраних товарів

Після натиснення на іконку кошика користувач переходить до сторінки кошика, де бачить всю потрібну інформацію про доданий товар та кінцеву вартість замовлення (рисунок 5.11). За потреби можна змінити кількість цього товару, тоді вартість перерахується і виведеться актуальна.

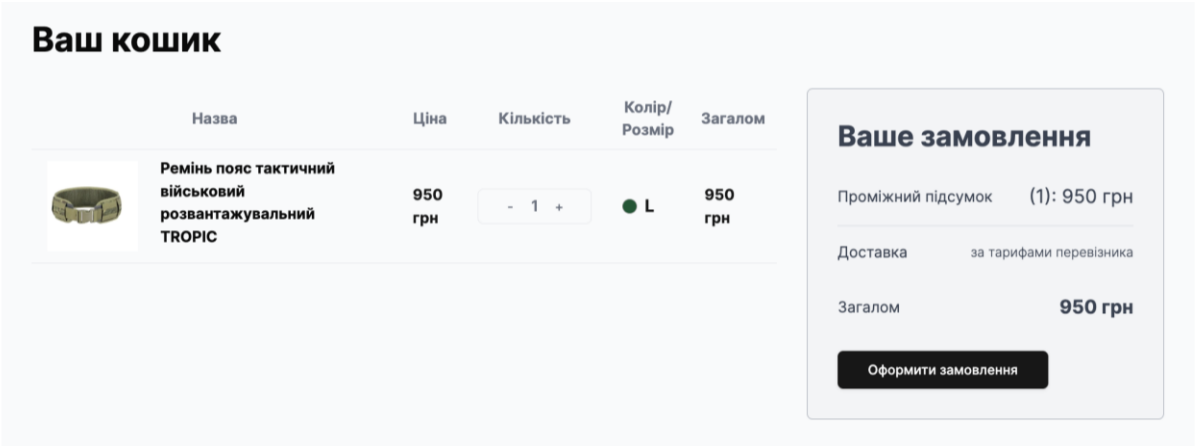


Рисунок 5.11 — Сторінка кошика

Наступним етапом, якщо користувач захоче оформити замовлення з доданими товарами він може натиснути на кнопку “Оформити замовлення”. Після натиснення потрапляємо на сторінку доставки, де вказуємо дані для доставки, а саме номер телефону, місто, та відділення пошти (рисунок 5.12).

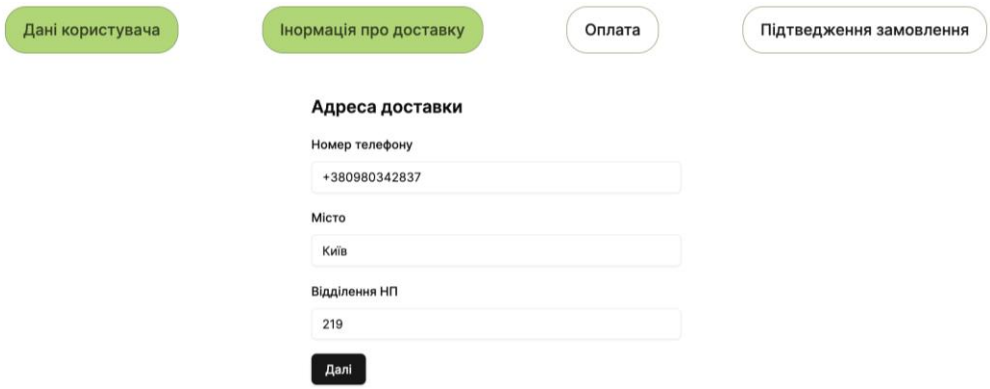


Рисунок 5.12 — Сторінка доставки

Зверху бачимо етапи оформлення замовлення. Зеленим підсвічуються пройдені етапи. Після підтвердження потрапляємо на сторінку оплати, де потрібно вибрати метод оплати та чи потрібно телефонувати менеджеру, щоб уточнити щось по замовленню. Наступним етапом перекидає на сторінку сформованого замовлення, там можна перевірити всю введену інформацію, за потреби її змінити (рисунок 5.13).

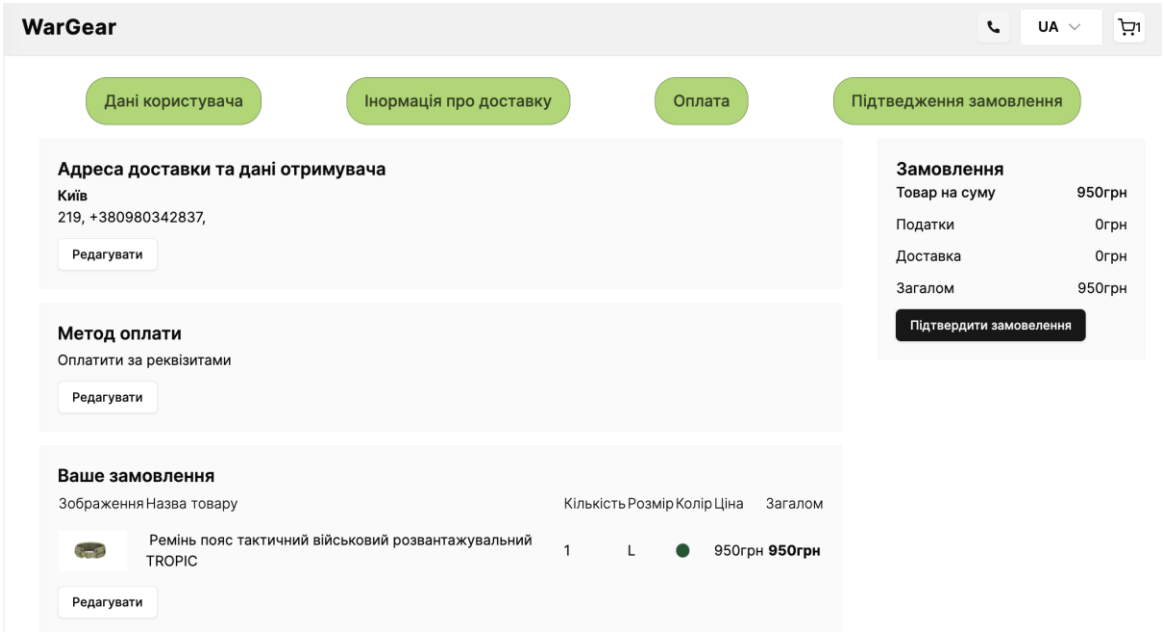


Рисунок 5.13 — Сторінка підтвердження замовлення

Якщо все вірно користувач натискає на кнопку підтвердження замовлення і тоді вся інформація про замовлення записується в базу даних.

За потреби користувач може перейти на сторінку історія замовлень, там він побачить таблицю зі всіма своїми замовленнями (рисунок 5.14).

Рисунок 5.14 — Історія замовлень

Натиснувши “Деталі”, потрапляємо до замовлення яке нас цікавить (рисунок 5.15).

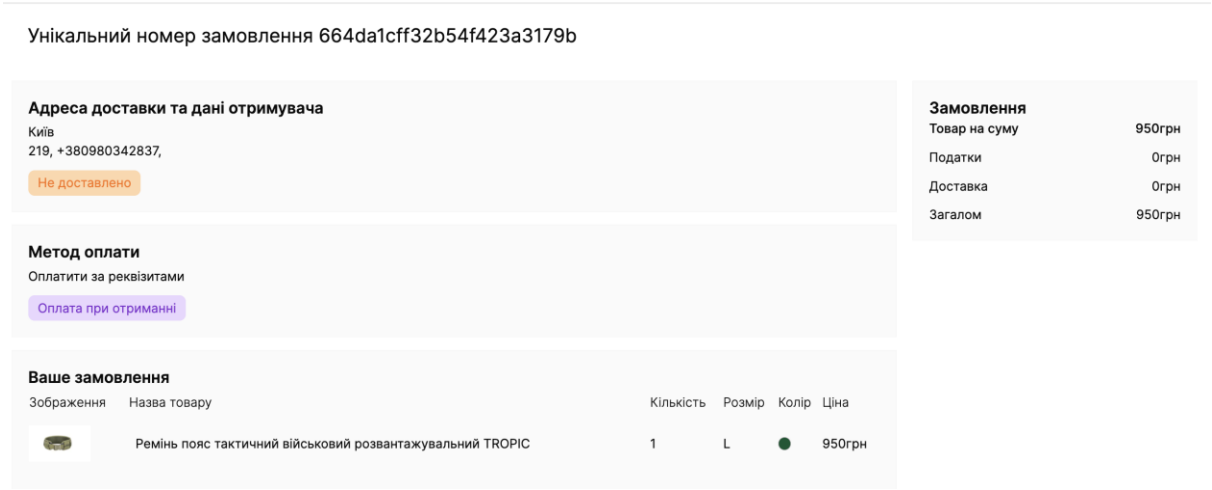


Рисунок 5.15 — Історія замовлень

На сторінці замовлення можемо побачити всю інформацію про замовлення та статус доставки.

Якщо ж користувач увійшов під акаунтом адміністратора, в нього в бічному меню з’являється розділ “Amin Dashboard” (рисунок 5.16).

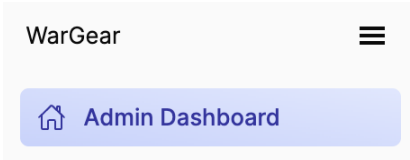


Рисунок 5.16 — Розділ Admin Dashboard

Перейшовши на цю сторінку, бачимо статистику веб-порталу, а саме на скільки грошей продано товару, кількість замовлень, кількість товарів та користувачів. Також присутні графіки статистики продажів та графік який показує співвідношення кількості товарів по категоріям (рисунок 5.17).

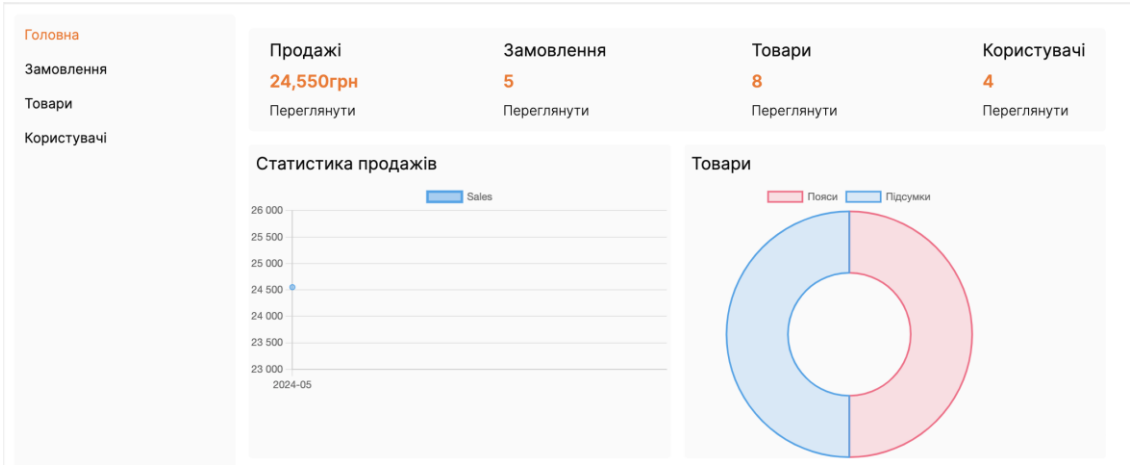


Рисунок 5.17 — Головна сторінка Admin Dashboard

На сторінці замовлення розміщується таблиця зі всіма замовленнями в магазині всіх користувачів, де також можна побачити деталі замовлення та адміністратор має можливість позначити замовлення як доставлене.

Перейшовши на сторінку товарів бачимо таблицю зі всіма товарами (рисунок 5.18).

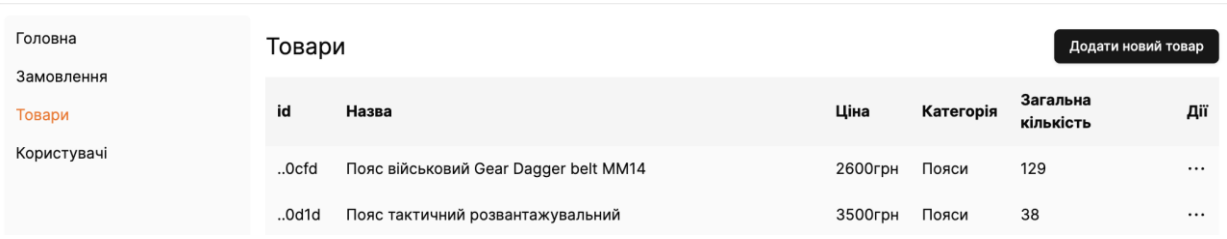


Рисунок 5.18 — Таблиця товарів

В діях до конкретного товару можна видалити його або змінити інформацію про товар. Натиснувши на кнопку “Додати новий товар”, додається товар з початковими тестовими даними, а після того можна змінити їх (рисунок 5.19).

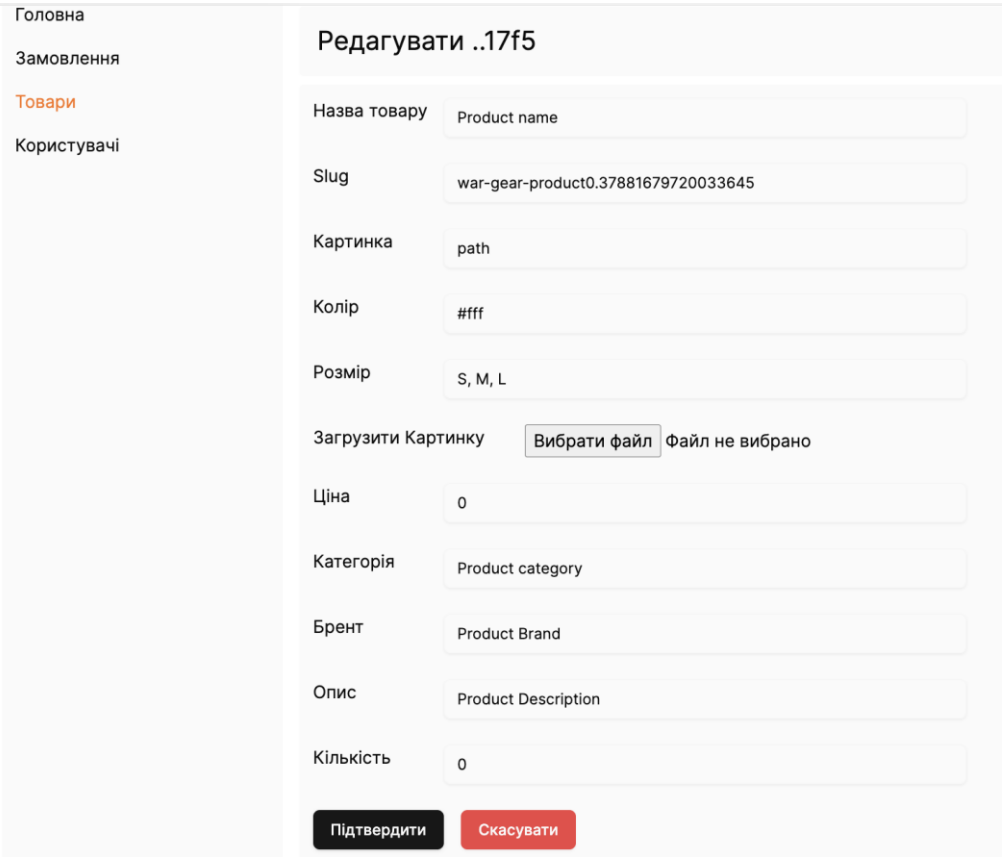


Рисунок 5.19 — Форма зміни даних про товар

На сторінці користувачів бачимо таблицю з користувачами, можна видалити конкретного користувача або призначити його адміністратором.

ВИСНОВКИ

В ході виконання роботи було розроблено веб-портал підбору військової амуніції та виконано наступні завдання.

1. Було проаналізовано існуючі веб-сервіси для підбору військової амуніції. В результаті було виділено їхні переваги та недоліки, які було враховано при розробці. Вияснили, що багато з існуючих сервісів мають застарілий дизайн, перенаповненість інформацією, застарілі технології та відсутність функціоналу, якого очікують користувачі від подібного веб-порталу. Було досліджено цільову аудиторію створеного веб-порталу, вона включає військовослужбовців, волонтерські організації, державні установи та цивільні люди. Проаналізувавши потреби користувачів було вияснено, що більше за все їм потрібен простий інтуїтивно зрозумілий інтерфейс, швидкодія, достатній функціонал, локалізація та зручна система пошуку і фільтрації.

2. Проаналізовано та підібрано засоби реалізації веб-сервісів, які були використані для розробки веб-порталу підбору військової амуніції. Розглянули особливості обраних технологій та переваги їхнього використання.

3. Розроблено архітектуру веб-порталу для підбору військової амуніції. Для розробки було обрано клієнт-серверну архітектуру враховуючи переваги такого підходу.

4. Написаний програмний код веб-порталу, а саме створено базу даних та моделі користувачів, товарів та замовлень. Розроблено API для взаємодії з базою даних, ендпоінти для керування товарами, користувачами і замовленнями. Розроблено інтерфейс користувача з можливістю реєстрації та авторизації, перегляду товарів, пошуку і фільтрації товарів, змінення інформації користувача на сторінці профілю, додавання товарів в кошик та до обраного, оформлення замовлення. Для адміністрування веб-порталу створено сторінки, які надають можливість додавання та редагування товарів, перегляду та керування всіх замовлень включаючи вказування успішності доставки, перегляду всіх користувачів з можливістю призначення ролі адміністратора. Також створені діаграми, які зображають статистику продажів.

5. Розроблене програмне забезпечення було комплексно протестовано, підтвердивши працездатність всіх функцій веб-порталу. Тестування підтвердило виконання всіх вимог до сервісу, зручність, ефективність та надійність.

Розроблений веб-портал може бути масштабований в майбутньому шляхом додавання різноманітних способів оплати замовлень, таких як LiqPay, PayPal та інших банківських систем. Також можливе додавання нового функціоналу, а саме рекомендації товарів на основі аналізу переглядів товарів користувачем та спеціалізовані пропозиції.

Програмне забезпечення може бути використано приватними підприємцями та виробниками військової амуніції з метою продажу. В майбутньому веб-портал може стати незамінним помічником для користувачів, які цінують якість, зручність та надійність.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мельник Р. А. Програмування веб-застосувань (фронт-енд та бек енд): навч.посіб. Львів: Видавництво Львівської політехніки, 2018. 248 с.

2. Інтернет-магазин військової амуніції PROF1GROUP. URL: https://prof1group.ua/ (дата звернення: 02.06.2024).

3. Розширення для тестування та аналізу веб-сайтів Lighthouse. URL: https://chrome.google.com/webstore/detail/lighthouse/blipmdconlkpinefehnmjammfjpmpbjk?hl (дата звернення: 02.06.2024).

4. Інтернет-магазин Tactical Gear. URL: https://tacticalgear.ua/ (дата звернення: 02.06.2024).

5. Jon Duckett. HTML & CSS: Design and Build Websites. Published by Wiley, 2011. 512 p.

6. Alex Banks, Eve Porcello. Learning React: Modern Patterns for Developing React Apps. Published by O'Reilly Media, 2020. 350 p.

7. Kirill Konshin. Next.js Quick Start Guide: Server-side rendering done right. Published by Packt Publishing, 2019. 186 p.

8. Next.js Documentation. URL: https://nextjs.org/docs (дата звернення: 02.06.2024).

9. Cherny B. Programming TypeScript: Making Your JavaScript Applications Scale. Published by O'Reilly Media, 2019. 322 p.

10. MongoDB Documentation. URL: https://docs.mongodb.com (дата звернення: 02.06.2024).

11. Zustand Documentation. URL: https://docs.pmnd.rs/zustand/getting-started/introduction (дата звернення: 02.06.2024).

12. Tailwind CSS documentation. URL: https://tailwindcss.com/docs/ (дата звернення: 02.06.2024).

13. Shadcn/ui. URL: https://ui.shadcn.com/ (дата звернення: 02.06.2024).

14. Шаховська Н. Б., Пасічник В. В. Сховища та простори даних: монографія. Львів: Видавництво Львівської політехніки, 2009. 244 с.

15. Ethan Brown. Web Development with Node and Express: Leveraging the JavaScript Stack. Published by O'Reilly Media, 2019. 332 p.

16. Eric Elliott. Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Modern JS Libraries. Published by O'Reilly Media, 2014. 254 p.

17. David Flanagan. JavaScript: The Definitive Guide: Master the World’s Most-Used Programming Language 7th Edition. Published by O'Reilly Media, 2020. 706 p.

18. Santana Roldan C. React 18 Design Patterns and Best Practices. Published by Packt Publishing, 2023. 524 c.

ДОДАТОК А

ТЕКСТ ПРОГРАМНОГО МОДУЛЯ

«РЕАЛІЗАЦІЯ КОШИКА ТА ЕНДПОІНТА ДЛЯ ОТРИМАННЯ ТОВАРІВ З ФІЛЬТРАЦІЄЮ І ПОШУКОМ У ВЕБ-ПОРТАЛІ ПІДБОРУ ВІЙСЬКОВОЇ АМУНІЦІЇ»

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ІАТЕ_ЦТЕ_ТР-01_24Б

Аркушів 6

Київ-2024

```
import { create } from 'zustand'
import { round2 } from '../utils'
import { OrderItem, ShippingAddress } from '../models/OrderModel'
import { persist } from 'zustand/middleware'

type Cart = {
  items: OrderItem[]
  itemsPrice: number
  taxPrice: number
  shippingPrice: number
  totalPrice: number

  paymentMethod: string
  shippingAddress: ShippingAddress
}
const initialState: Cart = {
  items: [],
  itemsPrice: 0,
  taxPrice: 0,
  shippingPrice: 0,
  totalPrice: 0,
  paymentMethod: '',
  shippingAddress: {
    fullName: '',
    address: '',
    city: '',
  },
}
```

```
export const cartStore = create<Cart>()(
  persist(() => initialState, {
    name: 'cartStore',
  })
)

export default function useCartService() {
  const {
    items,
    itemsPrice,
    taxPrice,
    shippingPrice,
    totalPrice,
    paymentMethod,
    shippingAddress,
  } = cartStore()
  return {
    items,
    itemsPrice,
    taxPrice,
    shippingPrice,
    totalPrice,
    paymentMethod,
    shippingAddress,
    increase: (item: OrderItem) => {
      const exist = items.find(
        (x) =>
          x.slug === item.slug && x.color === item.color && x.size === item.size
      )
      let updatedCartItems = exist
        ? items.map((x) =>
```

```
        x.slug === item.slug && x.color === item.color && x.size === item.size
          ? { ...exist, qty: exist.qty + 1 }
          : x
      )
      : [...items, { ...item, qty: 1, color: item.color, size: item.size }]
    const { itemsPrice, shippingPrice, taxPrice, totalPrice } =
      calcPrice(updatedCartItems)
    cartStore.setState({
      items: updatedCartItems,
      itemsPrice,
      shippingPrice,
      taxPrice,
      totalPrice,
    })
  },
  decrease: (item: OrderItem) => {
    const exist = items.find((x) => x.slug === item.slug && x.color === item.color &&
x.size === item.size)
    if (!exist) return
    let updatedCartItems;

    if (exist.qty === 1) {
      updatedCartItems = items.filter((x) => !(x.slug === item.slug && x.color ===
item.color && x.size === item.size))
    } else {
      updatedCartItems = items.map((x) =>
        (x.slug === item.slug && x.color === item.color && x.size === item.size) ? {
...x, qty: Math.max(1, x.qty - 1) } : x
      )
    }
    const { itemsPrice, shippingPrice, taxPrice, totalPrice } =
```

```
        calcPrice(updatedCartItems)
      cartStore.setState({
        items: updatedCartItems,
        itemsPrice,
        shippingPrice,
        taxPrice,
        totalPrice,
      })
    },
    saveShippingAddrress: (shippingAddress: ShippingAddress) => {
      cartStore.setState({
        shippingAddress,
      })
    },
    savePaymentMethod: (paymentMethod: string) => {
      cartStore.setState({
        paymentMethod,
      })
    },
    clear: () => {
      cartStore.setState({
        items: [],
      })
    },
    init: () => cartStore.setState(initialState),
  }
}

const calcPrice = (items: OrderItem[]) => {
  const itemsPrice = round2(
      items.reduce((acc, item) => acc + item.price * item.qty, 0)
```

```
    ),
    shippingPrice = 0,
    taxPrice = 0,
    totalPrice = round2(itemsPrice + shippingPrice + taxPrice)
  return { itemsPrice, shippingPrice, taxPrice, totalPrice }
}
```

Реалізація ендпоінта для отримання товарів з фільтрацією і пошуком

```
import dbConnect from '@/lib/dbConnect'
import ProductModel from "@/lib/models/ProductModel";
import { NextRequest } from "next/server";

interface Filter {
  name: {
    $regex: string;
    $options: string;
  };
  price?: {
    $gte?: number;
    $lte?: number;
  };
  colors?: {
    $regex: string;
    $options: string;
  };
  sizes?: {
    $regex: string;
    $options: string;
  }
}
```

```
export const GET = async (req: NextRequest) => {
  await dbConnect()
  const url = new URL(req.url)
  const search = url.searchParams.get("search")
  let filter: Filter = { name: { $regex: `${search}`, $options: 'i' } }

  const lte = url.searchParams.get("lte")
  const gte = url.searchParams.get("gte")
  const color = `${url.searchParams.get("color")}`
  const size = `${url.searchParams.get("size")}`

  if (size !== null && size !== '') {
    filter.sizes = { $regex: `${size}`, $options: 'i' }
  }

  if (color !== null && color !== '') {
    filter.colors = { $regex: `#${color}`, $options: 'i' }
  }

  if (lte !== null && !isNaN(parseFloat(lte))) {
    filter.price = { ...filter.price, $lte: parseFloat(lte) }
  }

  if (gte !== null && !isNaN(parseFloat(gte))) {
    filter.price = { ...filter.price, $gte: parseFloat(gte) }
  }
  const products = await ProductModel.find(filter).lean();
  return Response.json(products)
}
```