Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Фоменка Владислава Олександровича*
(ПІБ)

академічної групи *121-20-2*
(шифр)

спеціальності *121 «Інженерія програмного забезпечення»*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка веб-додатку для інтернет магазину одягу з автоматизованою системою прийому замовлень*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*проф. Мороз Б.І.*			
розділів:				
спеціальний	*проф. Мороз Б.І.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Мартиненко А.А*			

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

(підпис) (прізвище, ініціали)

« » 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу
бакалавра
(назва освітньо-кваліфікаційного рівня)

студентки *121-20-2* *Фоменка Владислав Олександрович*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка веб-додатку для інтернет магазину одягу з автоматизованою системою прийому замовлень*

затверджена наказом ректора НТУ «ДП» від *23.05.2024* № *470-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів практик та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***01.06.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***06.06.2024 р.***

Завдання видав _____ *проф. Мороз Б.І.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Фоменко В.О.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.24 р.*

Термін подання кваліфікаційної роботи до ЕК: *10.06.24 р.*

РЕФЕРАТ

Пояснювальна записка: 69 с., 34 рис., 3 дод., 20 джерел.

Об’єкт розробки: веб-додаток для інтернет магазину одягу з автоматизованою системою прийому замовлень

Мета кваліфікаційної роботи: розробити веб додаток для інтернет магазину одягу з автоматичним прийомом замовлень, для подальшого адміністрування ними.

У вступі описана актуальність проблеми та її аналіз в новітніх реаліях. Визначається основний напрямок розробки та мета кваліфікаційної роботи.

У першому розділі проаналізовано предметну область, актуальність проблеми та мету розробки, визначено вимоги, програмні засоби та методи реалізації програмного забезпечення.

У другому розділі обґрунтовано вибір платформи розробки, спроектовано та розроблено програмне забезпечення, описано роботу програми, її структуру та функціональні алгоритми, а також виклик та завантаження програми, описано використані технічні засоби.

В економічній частині було визначено трудомісткість розробки програмного забезпечення, розраховано вартість створення програмного забезпечення та визначено час, необхідний для його розробки.

Практичне завдання погягає створені веб-додатку, що забезпечує ефективку роботу інтернет-магазину з його користувачами, дає змогу клієнтам зробити вибір серед товарів та оформити замовлення, в той час як адміністратору зручно переглядати прийняті замовлення.

Розробка веб-додатку для інтернет-магазину є актуальною для українського ринку, особливо в умовах війни, може допомогти компаніям залучити нових клієнтів та розширити свою аудиторію за рахунок використання інструментів цифрового маркетингу.

Список ключових слів: ІНТЕРНЕТ-МАГАЗИН, ВЕБ-ДОДАТОК, VUE.JS, JAVASCRIPT, NODEJS, EXPRESS.JS, HTML, CSS, TAILWIND

ABSTRACT

Explanatory note: 69 p., 34 figures, 3 apps, 20 sources.

Object of development: web application for an online clothing store with an automated order acceptance system

The purpose of the qualification work: to develop a web application for an online clothing store with an automated order acceptance system for further administration.

The introduction describes the relevance of the problem and its analysis in the modern realities. The main direction of development and the purpose of the qualification work are defined.

The first section analyzes the subject area, the relevance of the problem and the purpose of development, defines the requirements, software tools and methods of software implementation.

The second section justifies the choice of the development platform, designs and develops the software, describes the operation of the program, its structure and functional algorithms, as well as the program call and download, and describes the technical means used.

In the economic part, the labor intensity of software development was determined, the cost of creating the software was calculated, and the time required for its development was determined.

The practical task involves the creation of a web application that ensures the efficient operation of an online store with its users, allows customers to make a choice among the products and place an order, while the administrator can conveniently view the accepted orders.

The development of a web application for an online store is relevant for the Ukrainian market, especially in times of war, and can help companies attract new customers and expand their audience through the use of digital marketing tools.

List of keywords: ONLINE STORE, WEB APPLICATION, VUE.JS, JAVASCRIPT, NODEJS, EXPRESS.JS, HTML, CSS, TAILWIND

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних

JS – Java Script

ІТ – інформаційні технології

HTML – Hypertext Markup Language

CSS – Cascading Style Sheets

CMS - Система управління контентом

ПК – Персональний комп’ютер

ІС – Інформаційна система

DOM – об’єктна модель документа.

ВСТУП

Розширення горизонтів продажів стає необхідністю для зростаючого підприємства. В сучасному світі, коли традиційні фізичні магазини все більше зачиняються, а продажі в онлайн сфері зростають, перехід до електронної комерції стратегічним кроком для бізнесу.

За оцінкою EVO за 2023 рік частка електронної комерції в Україні виросла с 9% до 11-12%, що становить понад 10 мільйонів інтернет покупців. Це свідчить що ринок став більше аніж в довоєнний час та є тенденція на його зростання.

Інтернет магазини є ключовим елементом для створення онлайн бізнесу, відсутність якого унеможливлює подальший розвиток підприємства. Веб додатки надають в повній мірі всю інформацію про конкретний певний товар, зацікавлюють потенційного покупця зробити покупку, та також мають переваги для зручності організації робочого процесу для адміністрування замовленнями.

Однією із самих важливих інтернет-магазинів є автоматизація більшості бізнес-процесів, що скорочує витрати та підвищує ефективність роботи підприємства.

Задачею кваліфікаційної роботи є створення веб-додатку інтернет-магазину що допоможе полегшити роботу із замовленнями та розробити зручний сервіс для клієнтів.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі.

Зараз не всім зручно відвідувати звичайні магазини. Підприємцям також важко приймати фінансові ризики, пов'язані з відкриттям та утриманням фізичних магазинів. Більш доцільним є створення веб додатку, який буде мати переваги для бізнесу та кінцевих покупців.

Веб додаток для інтернет магазину потрібен для спрямування на нього трафіку з інших інтернет ресурсів, а сам веб додаток насамперед забезпечить функцію прийому замовлень від клієнтів.

Інтернет-магазини дозволяють підприємствам більш ефективно управляти своїми ресурсами. Автоматизовані системи зменшують потребі в найму персоналу для прийому та обробки замовлень.

Основними частинами веб додатку для електронної комерції є:

1. Frond-end

Ця частина для взаємодіє з користувачами додатку і включає:

- Перелік товарів, де відображено фото товару, його ціна, опис
- Сторінка для кожного товару
- Кошик
- Фільтрація, сортування
- Замовлення

2. Back-end

Серверна частина, що включає:

- База даних
- Система обробки замовлень
- Система управлення користувачами

3. Адміністративна панель

- Управління контентом

- Перегляд замовлень

1.2. Призначення розробки та галузь застосування.

Метою кваліфікаційної роботи є створення працюючого веб-додатку для магазину одягу підприємства “deadmarket.ua” який буде виконувати потреби бізнесу та клієнтів. Система повинна в першу чергу полегшити прийом замовлень, та забезпечити користувачам легкість в користуванні. Перелік товару буде відображений на головній сторінці сайту де можна буде зробити фільтр по категоріям. Замовлення які будуть оформлені покупцем можна буде переглянути в адміністративній панелі.

1.3. Підстави для розробки.

Підставою для розробки є завдання “Розробка веб-додатку для інтернет магазину одягу з автоматизованою системою прийому замовлень.” На виконання кваліфікаційної роботи затверджене кафедрою Програмного забезпечення комп’ютерних систем Національного технічного університету “Дніпровська політехніка”

1.4. Постановка завдання

Треба розробити веб-додаток для інтернет-магазину одягу та взуття.
Додаток має задовільнити наступні критерії:

- Інтуїтивний та доступний інтерфейс: Додаток має легко сприйматись користувачами будь якого рівня навичок коритування інтернетом. Всі основні елементи, такі як меню, кнопки, кошик, мають бути розміщені логічно, щоб можна було знайти швидко необхідний функціонал та інформацію
- Зручність в користуванні: Додаток повинен забезпечити зручність під

час виконання користувачем будь яких взаємодій з ним, швидко завантажувати сторінки та обробляти інформацію, і мати мінімальну кількість кроків до здійснення замовлення.

Щоб досягти поставлених цілей треба створити систему яка повинна включати в собі клієнтську частину, серверну частину та базу даних

1.5. Вимоги до програми або програмного виробу.

1.5.1. Вимоги до функціональних характеристик.

Система має виконувати наступні функції:

- Відображення товару:
- Сортування, фільтрація
- Прийом замовлень
- Зберігання отриманих даних
- Авторизація користувачів
- Відображення збережених даних

1.5.2. Вимоги до інформаційної безпеки.

Основна вимога до інформаційної безпеки додатку інтернет-магазину є конфіденційність інформації про користувачів

Для коректної роботи додатку треба реалізувати:

- перевірку семантики та синтаксису введених даних;
- виведення інформації про помилки
- безперервну роботу протягом не менше 120 годин (5 діб)
- незалежність від платформи

1.5.3. Вимоги до складу та параметрів технічних засобів.

Система не є вибагливою до ресурсів, тому для функціонування системи, потрібен стаціонарний комп’ютер, який буде відповідати наступним мінімальним вимогам:

- процесор з тактовою частотою 2.4 ГГц
- як мінімум 2 гігабайти оперативної пам’яті
- доступ в мережу Інтернет
- комп’ютерна миша
- клавіатура

Перечислені вище характеристики є мінімальними рекомендованими. Додаток буде розрахований на максимально велику кількість користувачів, тому при розробці врахована доступність для всіх систем

1.5.4. Вимоги до інформаційної та програмної сумісності.

Для того щоб можна було отримати доступ до інтернет-додатку, програмне забезпечення на мобільному пристрої чи на персональному комп’ютері повинно відповідати наступним видогам:

- Сучасний веб-браузер, такий як Google Chrome, Mozilla Firefox, Safari, Microsoft Edge
- Будь яка операційна система, наприклад Windows, MacOs, Android, IOS, Linux

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Веб-додаток інтернет магазину призначений для електронної комерції. Виходячи з цього система повинна мати наступний функціонал:

- Відображення товару на головній сторінці
- Сортування товару по ціні
- Фільтрація товару по категоріям (одяг, аксесуари, взуття)
- Додавання товарів в улюблене та можливість перейти сторінку товару з улюбленого
- Додавання товарів в кошик, перегляд товарів в ньому та оформлення замовлення
- Перегляд зроблених замовлень в особистому кабінеті користувача та можливість їх відмінити
- Авторизація та реєстрація
- Підтримка оплати онлайн

Для адміністратору:

- Додавання, редагування та видалення товарів
- Перегляд усіх замовлень та зміна їх статусу
- Видалення замовлень

Система повинна мати легкий в розумінні, стильний та мінімалістичний інтерфейс.

2.2. Опис застосованих математичних методів

При розробці даного веб-додатку не було використано математичних методів, оскільки при розробці не було потреби у їх використанні. Основна

логіка програми побудована на операціях з базою даних, обробці запитів та керуванні станом додатку.

2.3. Опис використаної архітектури та шаблонів проєктування

Архітектура

Архітектура проекту розділена на фронтенд і бекенд, використовуючи сучасні веб-технології та шаблони проектування.

1) Фронтенд
- Фреймворк: Vue.js
- Сторонні бібліотеки: Pinia (для керування станом), Firebase (для аутентифікації)
- Модулі та компоненти: Використання компонентного підходу для створення відокремлених частин інтерфейсу, таких як корзина, улюблені товари, профіль користувача та панель адміністратора.

2) Бекенд
- Фреймворк: Express.js
- База даних: MongoDB (через Mongoose ODM)
- Хмарні сервіси: Firebase Admin SDK (для перевірки ролей користувачів)
- Middleware: Multer (для завантаження файлів), CORS (для керування міждоменною політикою доступу)

Шаблон проектування - MVC (Model-View-Controller)

Model: Mongoose схеми для товарів та замовлень (Product, Order)

View: Vue.js компоненти для відображення даних та взаємодії з користувачем

Controller: Express.js маршрути для обробки запитів та взаємодії з базою даних

2.4. Опис використаних технологій та мов програмування.

Щоб реалізувати даних веб-додаток було використано наступні мови програмування та технології:

- JavaScript: основна мова програмування додатку. Забезпечує інтерактивність на стороні клієнта та функціональність на стороні сервера.
- Бібліотека Vue.js: просунутий фреймворк JavaScript для створення користувацьких інтерфейсів. Vue.js забезпечує адаптивний і компонентний підхід до побудови інтерфейсу.
- NodeJS: серверне виконання JavaScript
- Express.js: веб-фреймворк для Node.js, який забезпечує простий спосіб створення серверних додатків та API. Express.js дозволяє швидко створювати маршрути для обробки HTTP-запитів, керувати сеансами користувачів та інтегруватися з різними базами даних.
- Tailwind CSS: Утилітарний CSS фреймворк, який дозволяє швидко створювати сучасні та адаптивні інтерфейси за допомогою готових CSS-класів. Tailwind CSS спрощує процес стилізації, дозволяючи розробникам зосередитися на функціональності та дизайні додатку.
- Axios API: бібліотека для виконання HTTP-запитів з браузера та Node.js
- MongoDB, Mongoose: MongoDB — це база даних NoSQL, яка забезпечує високу продуктивність та масштабованість для зберігання даних. Mongoose — це менеджер для MongoDB, який забезпечує взаємодію з базою даних, спрощуючи роботу з даними у Node.js.
- Firebase Auth, Firebase Admin SDK: Firebase Auth використовується для аутентифікації користувачів, забезпечуючи просту інтеграцію з різними методами входу (електронна пошта, Google, Facebook тощо). Firebase Admin SDK використовується для керування Firebase сервісами з бекенд-коду, включаючи перевірку токенів користувачів для аутентифікації та авторизації.

- Pinia: бібліотека для зберігання локальних даних

Переліченим вище технологіям та мовам програмування була надана перевага з наступних причин:

Vue.js: Клієнт розроблений на JavaScript із використанням фреймворку Vue.js для розробки Single-page application (SPA). Це веб-додаток, який працює всередині єдиної HTML-сторінки. Це означає, що під час навігації між різними частинами додатку не відбувається перезавантаження всієї сторінки, а лише динамічно змінюється її вміст. Vue.js має досить простий синтаксис, томі цей фреймворк полегшує розробку додатків в яких є потрібність в масштабуванні.

Для зберігання даних в localStorage була використана бібліотека Pinia, задля зберігання даних про вхід користувача, даних про товари в кошику та в розділі улюблені. Ця бібліотека досить легко налаштовується і має підтримку для Vue.js

На серверній стороні використовується бібліотека Express.js на платформі Node.js. Запити до сервера повертають дані у форматі JSON, які потім відображаються на клієнті за допомогою бібліотеки axios. Express.js дозволяє легко та швидко створювати веб-сервери та API, що скорочує час розробки. Легко налаштовувати та запускати веб-сервери для обробки HTTP-запитів (GET, POST, PUT, DELETE).

Firebase Auth була використана щоб полегшити розробку додатку. За допомогою Firebase Auth була реалізована авторизація та реєстрація користувачів. А за допомогою Firebase Admin SDK були захищені запит на сервер. Перевіряючи чи є користувач адміністратором по електронній пошті.

2.5. Опис структури програми та алгоритмів її функціонування.

Програма поділяється на три основні частини: клієнтську частину (frontend), серверну частину (backend) та базу даних. Кожна частина виконує свої специфічні функції та взаємодіє з іншими частинами через визначені інтерфейси.

Клієнт розділений по логічним частинам.

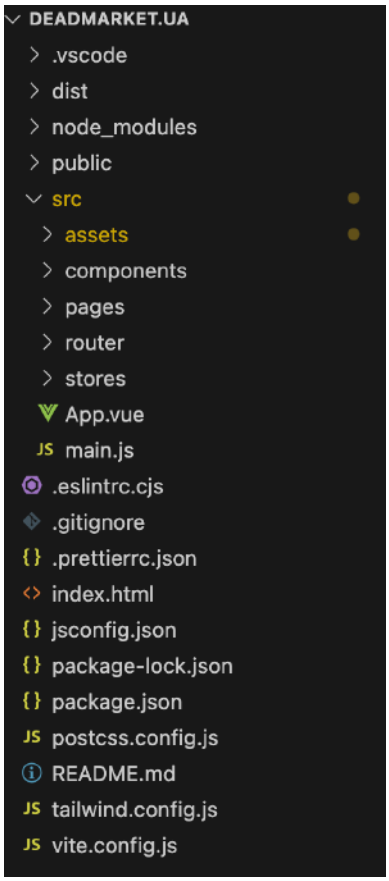

Рис. 2.1. Структура клієнтської частини

Головними файлами є App.vue, main.js та вміст папки src.

Розглянемо папку “components”. Компоненти сприяють кращій організації коду, розбиваючи додаток на логічні частини. Це полегшує розуміння та навігацію по коду, особливо в великих проектах. Кожен компонент це самодостатній блок коду, який можна повторно використовувати в різних частинах програми. Також Vue.js оптимізує рендеринг компонентів, оновлюючи лише ті частини DOM, які змінилися. Це підвищує продуктивність додатків, особливо при роботі з великими даними та складними інтерфейсами.

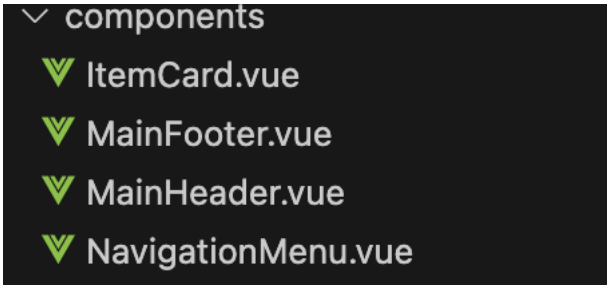

Рис. 2.2. Компоненти

В папкі “components” знаходится ItemCard.vue. Цей компонент відображає картинку продукту з цього ціною. Через велику кількість товарів цей компонент було логічно створити, задля оптимізації, полегшення розробки та масштабованості проекту.

“MainHeader.vue” є компонентом тегу HTML “<header></header>”. Він містить такі навігаційні елементи, як перехід у профіль, улюблені товари і перехід на сторінку кошику. Цей компонент присутній на всіх сторінках додатку.

“NavigationMenu.vue” містить компонент який відображається на головній сторінці “HomePage.vue”. Через великий об’єм домашної сторінки було прийнято рішення винести деяку її частину в окремий логічній компонент. Він містить елементи фільтрації та сортування товарів

“MainFooter.vue” компонет <footer></footer>. Має ссилки на сторінки “Про нас”,”Зв’язатись з нами”,”Повернення”

В папкі “pages” зберігаються наступні сторінки:

- AboutUs.vue – містить інформацію про магазин
- AdminEdit.vue – це сторінка для адміністратора інтернет магазину, яка дозволяє додавати товари, інформацію про них та картинки. Доступ до цьої сторінки має тільки адміністратор бо вона захищена за допомогою Firebase Admin SDK.
- AdminOrders.vue – це також сторінка до якої має доступ адміністратор. Вона відображає всю інформацію по зробленим замовленням. Після входу в свій аккаунт адміністратор потрапляє саме на цю сторінку.

- CartPage.vue – ця сторінка представляє кошик користувача. Вона відображає всі товари, додані до кошика, їх кількість та загальну суму до оплати. Користувач може видаляти товари з кошика, а також перейти до оформлення замовлення.
- ContactUs.vue – має форму для відправлення повідомлення
- HomePage.vue – головна сторінка веб-сайту. Вона відображає загальний вигляд інтернет-магазину з переліком категорій товарів. Ця сторінка є першою, яку бачить користувач при вході на сайт.
- LoginProfile.vue – сторінка для входу користувачів. Вона дозволяє користувачам увійти до свого профілю або зареєструватися, якщо у них ще немає облікового запису. Після успішного входу користувач потрапляє на сторінку свого профілю.
- ProductDetail.vue – сторінка з детальною інформацією про товар. Вона відображає всі характеристики товару, його зображення, доступні розміри, ціну та відгуки користувачів. Також тут користувач може додати товар до кошика або до улюблених.
- ProfilePage.vue – сторінка профілю користувача. Тут відображається особиста інформація користувача, історія його замовлень, а також можливість змінити особисті дані, такі як ім'я, прізвище, телефон та адресу доставки. Користувач також може скасувати замовлення, якщо воно ще не було відправлено.
- ReturnPage.vue – містить всю інформацію про повернення товару
- Index.js та App.vue є головними очками входу в клієнтську частину додатку

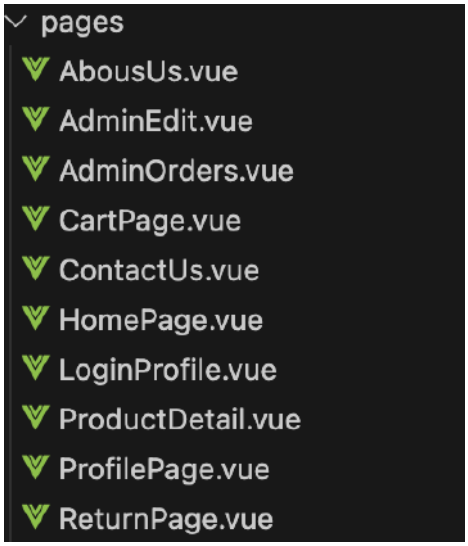

Рис 2.3. Сторінки

Рис. 2.4. Директива серверу

Директива серверу (рис 2.4) складається із наступних файлів та папок:

- Index.js – головний файл входу у весь додаток. Тут знаходиться головний код Node.js із логікою запитів та їх маршрутів.

- Uploads – тут знаходятся всі завантажені зображення товарів через Multer.

- Public – директива для зібраної клієнтської частини додатку.

Таким чином, клієнтська частина додатку розділена на різні модулі. Кожен файл має власну логіку то функціональність, що полегшує розробку то внесення змін. Серверна частина яка розміщена на віддаленому

серверному обладнанні складається із зібраного клієнту та файлів з кодом Node.js.

2.6. Обґрунтування та організація вхідних та вихідних даних програми

Веб-додаток взаємодіє з користувачем та його пристроєм отримуючи вхідні дані за допомогою текстового вводу, локального сховища та бази даних. Вхідні дані користувача отримуються через спеціальні форми. Наприклад на сторінці входу та реєстрації є форми для вводу пошти та паролю (рис 2.5).

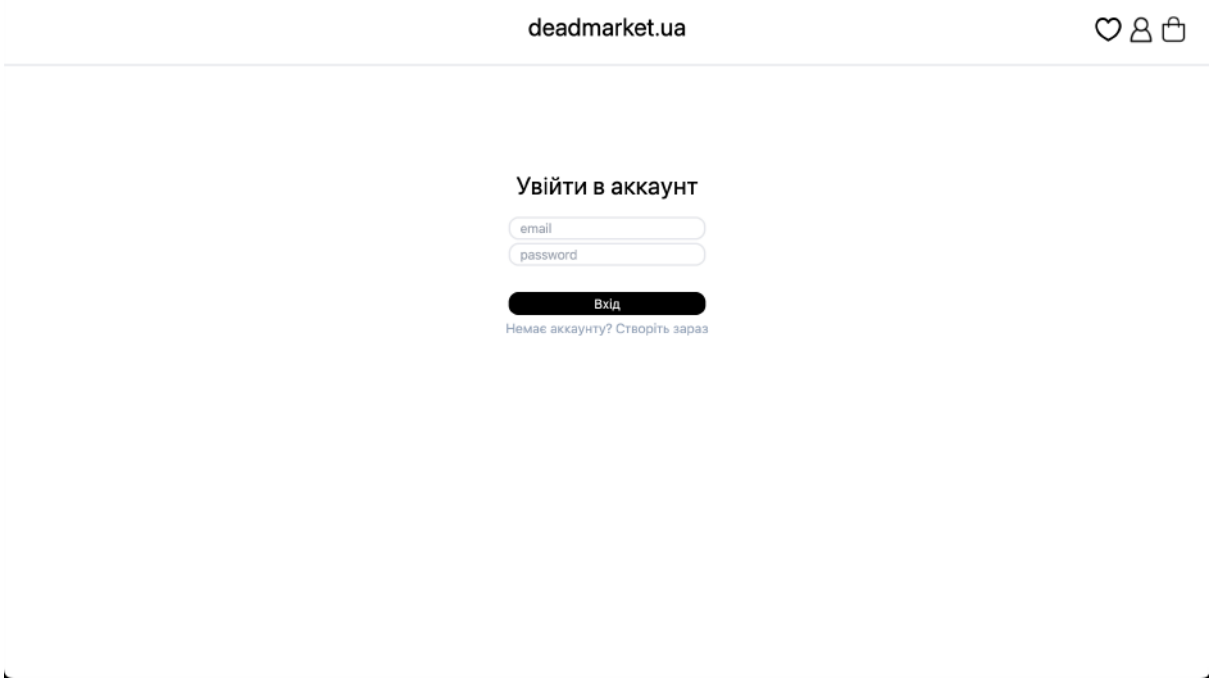

Рис 2.5. Сторінка входу

Після додавання товару в кошик з’являються форми даних для оформення замовлення (рис 2.6).

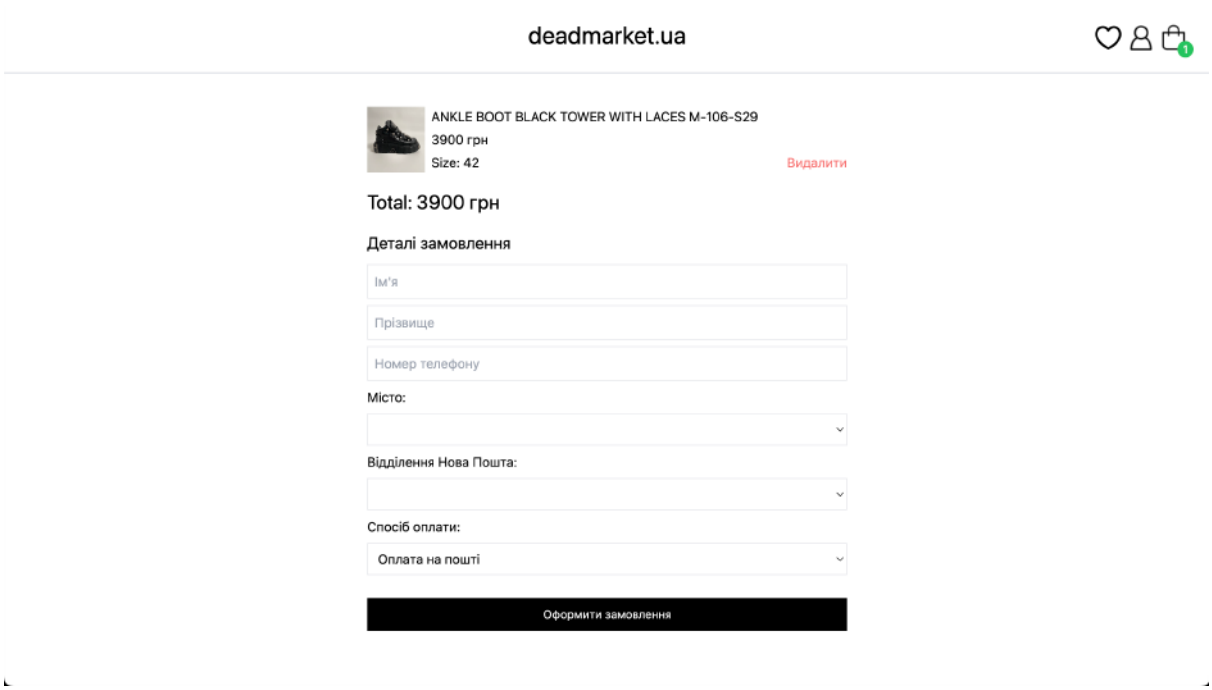

Рис 2.6. Сторінка кошику

Також у користувача адміністратор є доступ к сторінці редагування де окрім форм є спосіб додання зображення (рис 2.7).

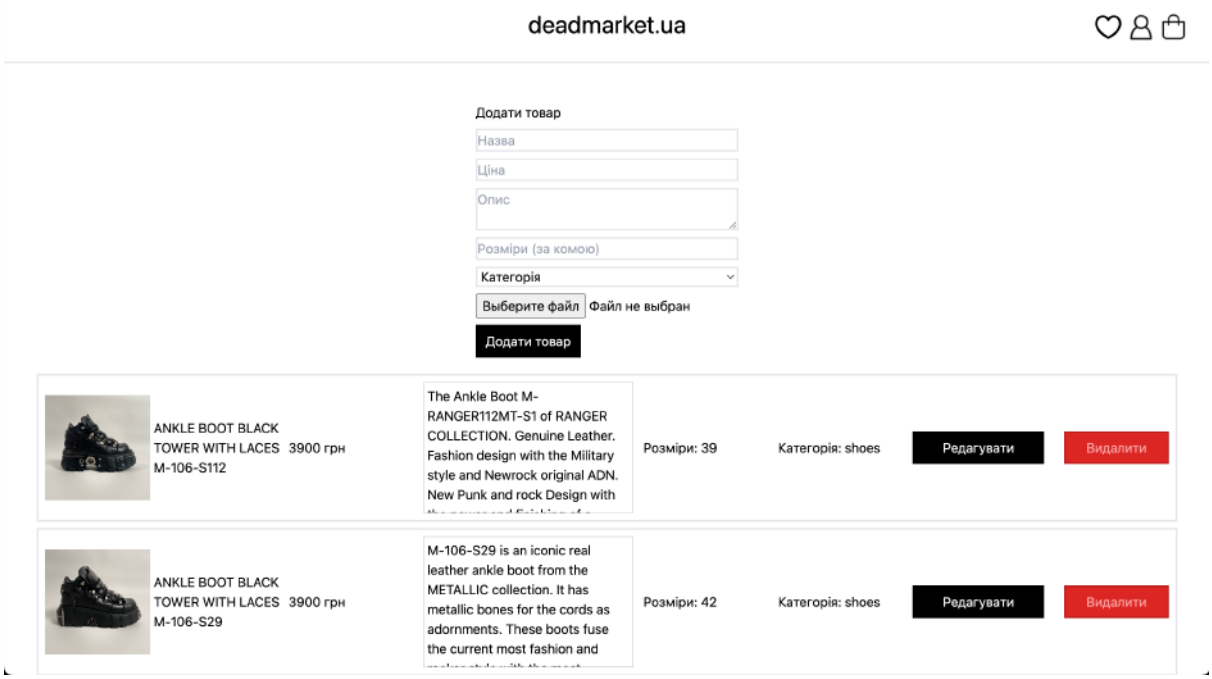

Рис 2.7. Сторінка редагування товарів

2.7. Опис роботи розробленої системи

2.7.1. Використані технічні засоби

Веб-додаток був розроблений та протестований на персональному ком’ютері з наступними характеристинами:

- Процесор: AMD Ryzen 7 2700x 3.4 GHz/32MB sAM4
- Материнська плата: MSI B450-A PRO MAX
- Оперативна пам’ять: HyperX 2*8 GB DDR4 3333 MHz
- Сховище даних: SSD 1 TB
- Монітор: Euromedia i27
- Клавіатура
- Комп'ютерна миша

2.7.2. Використані програмні засоби

Для розробти веб додатку було використане наступне програмне забезпечення:

- Visual Studio Code
- Google Chrome
- Figma
- Postman
- Commander One

Visual Studio Code була використана як основна IDE для написання коду та його тестування

Google Chrome використовувся як основний браузер для перегляду створеного сайту. Перевагою його використання були вбудовані інструменти розробника, які значно полегшують роботу з помилками

В Figma був зроблений макет сайту на основі якого був реалізовий остаточний вид додатку

Postman – програмне забезпечення для тестування API запитів на сервер

Commander One використовувався для з’єднання з сервером, для передачі даних

2.7.3. Виклик та завантаження програми

Розроблена система є веб-додатком. Це означає що його не потрібно завантажувати. Достатньо перейти за адресою “https://deadmarket.pp.ua/” і додаток сам завантажиться в браузері.

Додаток не займає багато оперативної пам’яті (рис 2.8)

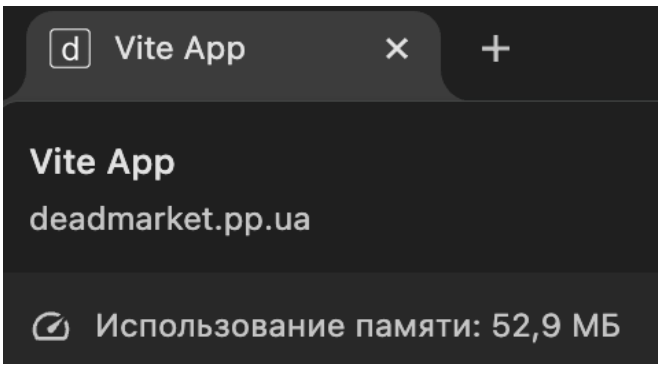

Рис. 2.8

2.7.4. Опис інтерфейсу користувача

Після того як користувач перейшов за адресою “https://deadmarket.pp.ua/” перед ним з’являється головна сторінка веб додатку (рис. 2.9). Тут відображені всі товари які є в наяності.

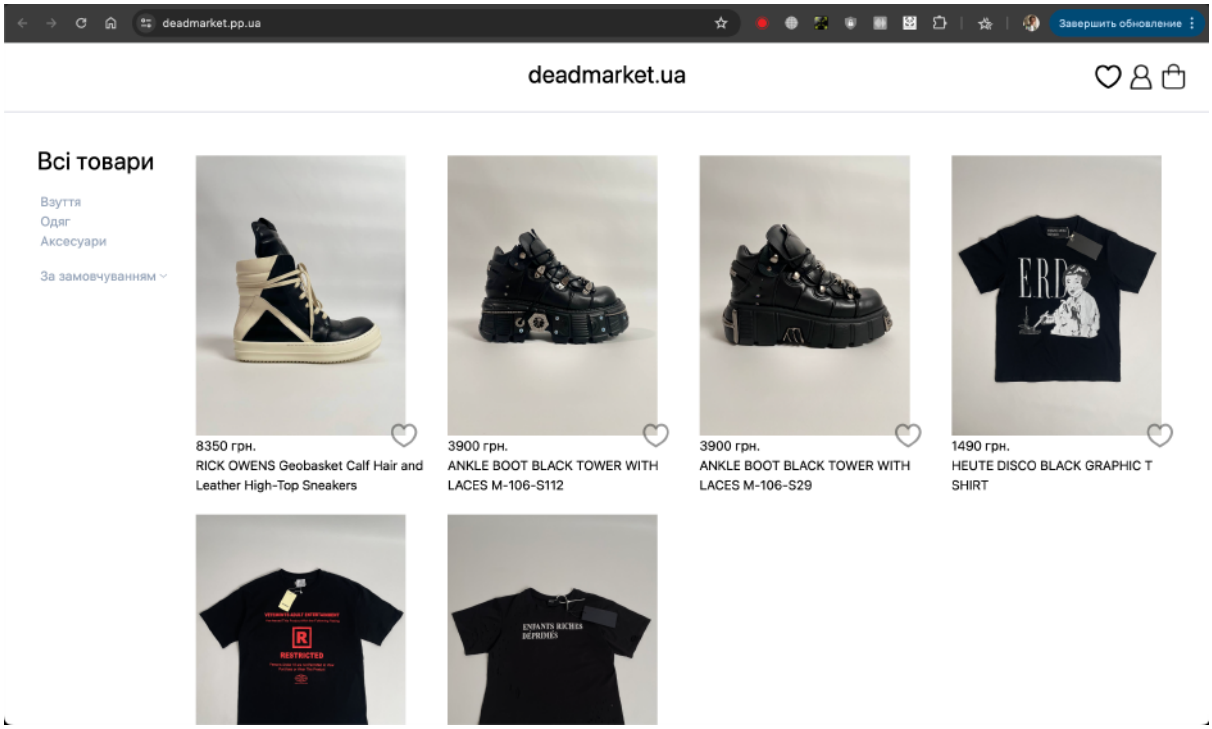

Рис. 2.9. Головна сторінка

На головній сторінці є фільтрація товару по категоріям. Щоб виконати фільтрацію, достатньо натиснути на категорію товару в меню зліва

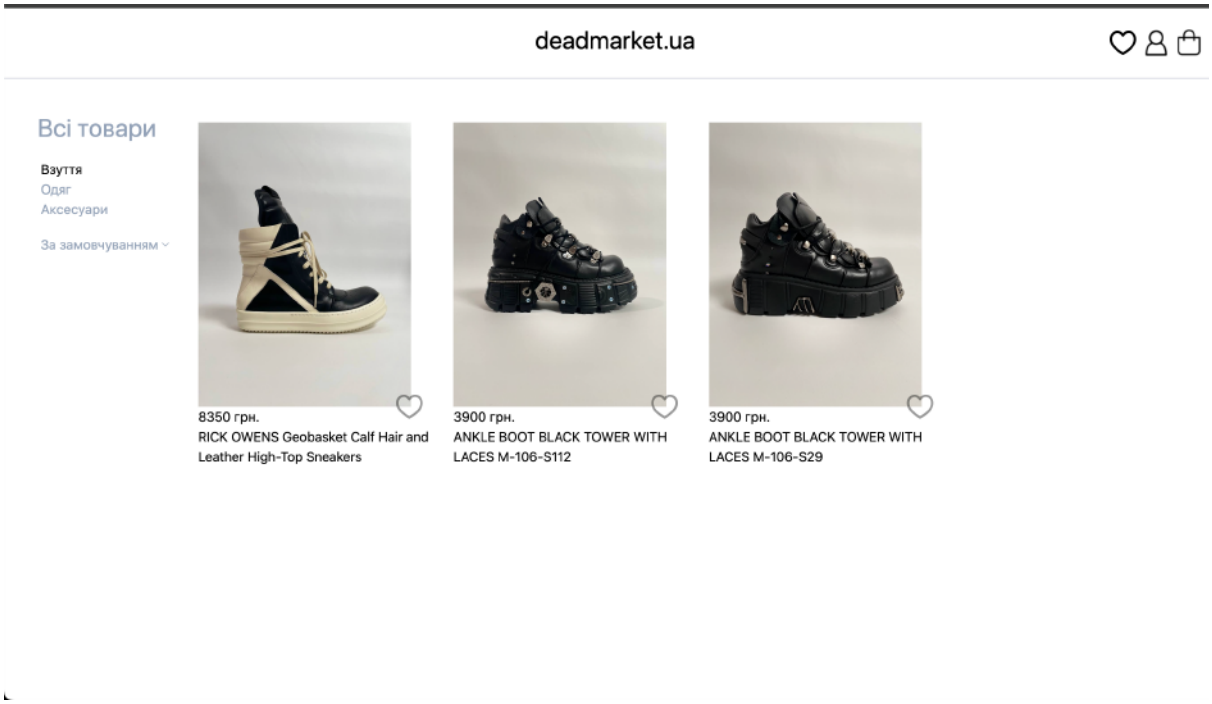

Рис. 2.10. Фільтрація

Також є сортування по ціні. Можна вибрати найдешевші або найдорожчі товари

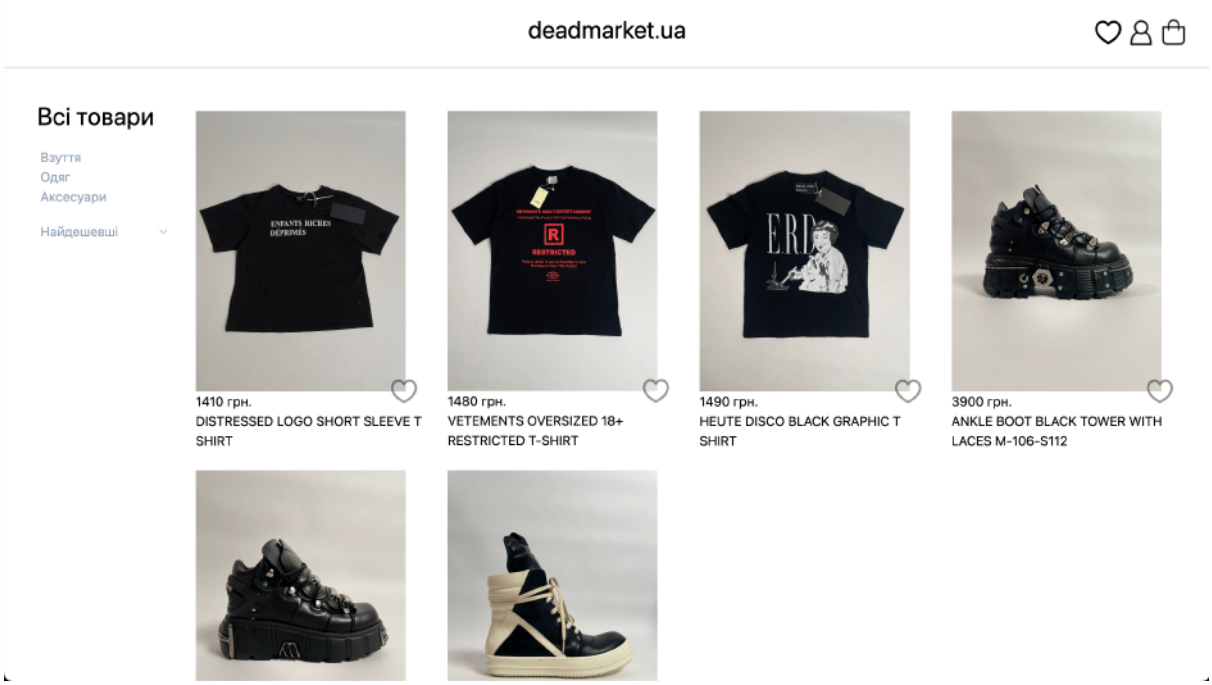

Рис. 2.11 – Сортування

Для зручності в орієнтуванні товарів є функція улюблене. При натисненні відповідної кнопки біля товару можна додати товар улюблені товари.

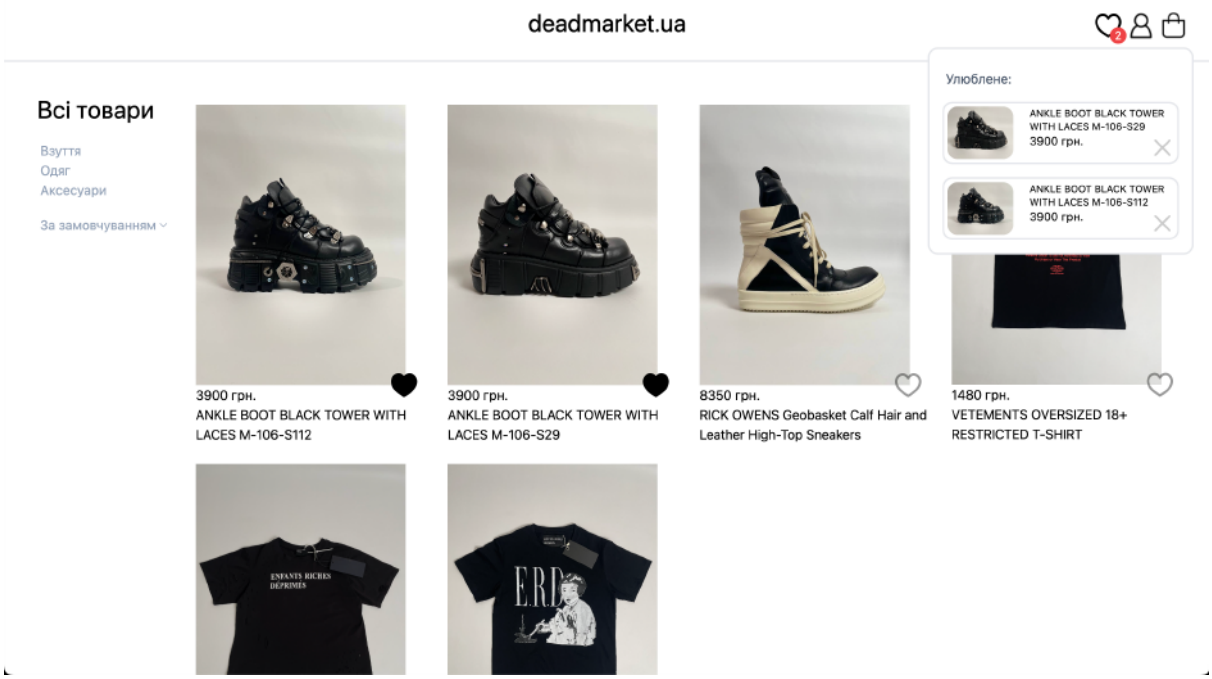

Рис. 2.12. Функція “Улюбленне”

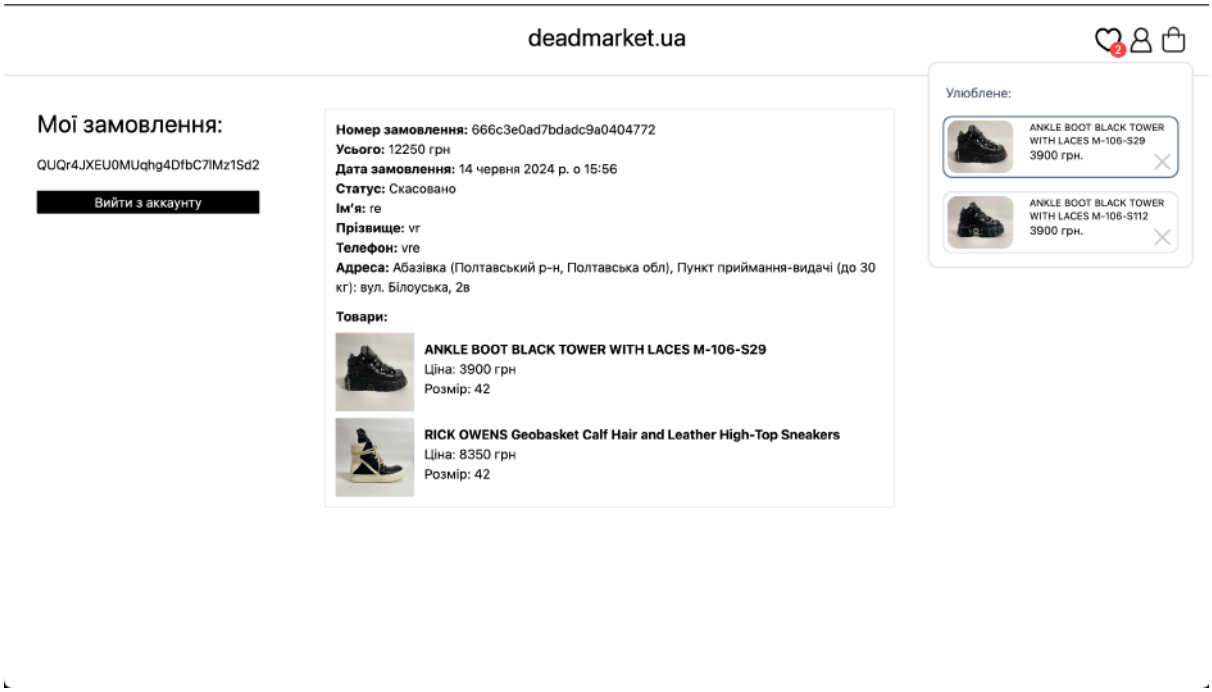

Рис. 2.13. Демонстарція роботи “Улюблене” з іншої сторінки

Отримати доступ до своїх улюблених товарів можна з будь якої частини сайту (рис 2.13)

При натиснені картки товару відбувається маршрутизація по даному товару, де відобрається інформація по ньому та є можливість додати його до кошику

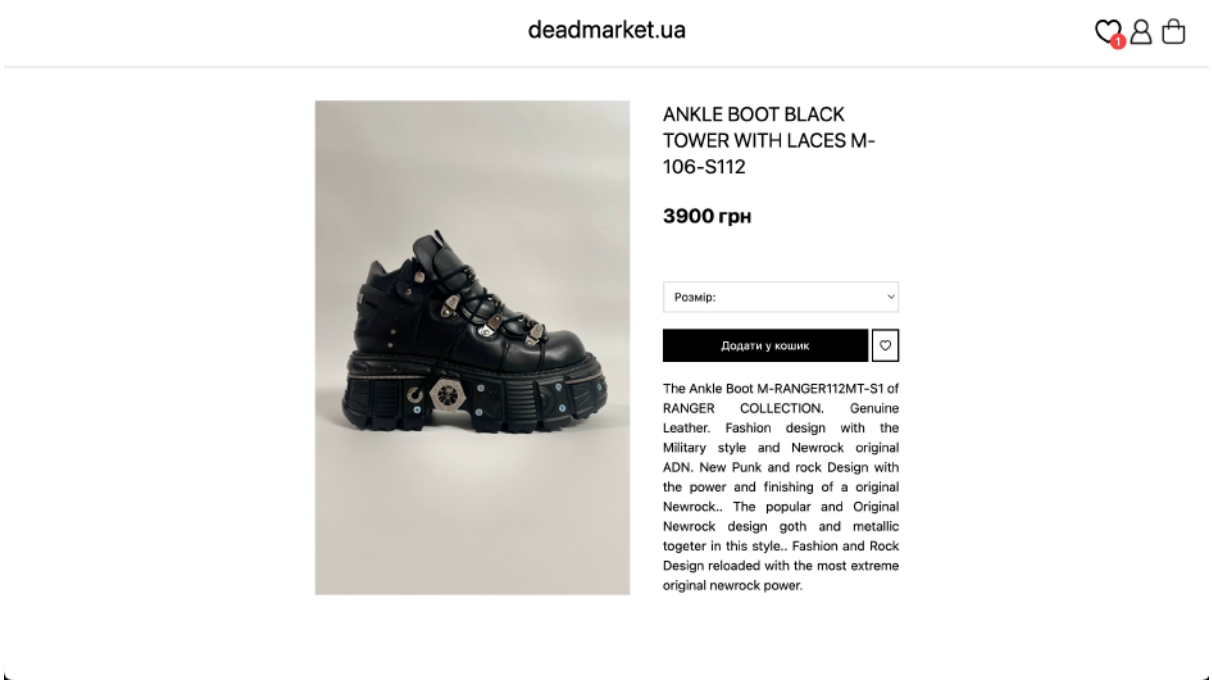

Рис. 2.14. Сторінка товару

При виборі розміру відкивається випадний список. Також на цій сторінці є можливість додати або видалити товар з улюбленого

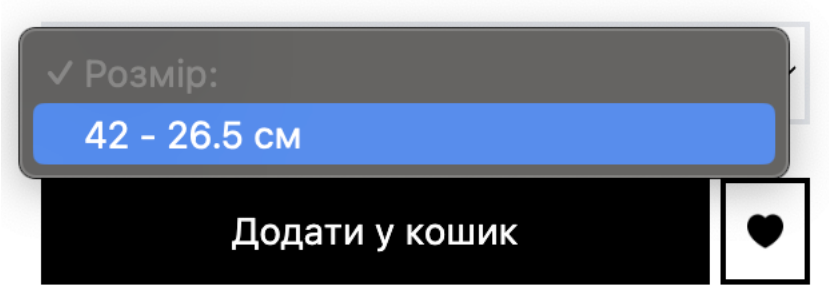

Рис. 2.15. Вибір розміру

Була проведена робота адаптації для мобільних пристоїв, тому на дисплеях різного формату інтерфейс має гарний вигляд та правильне відображення

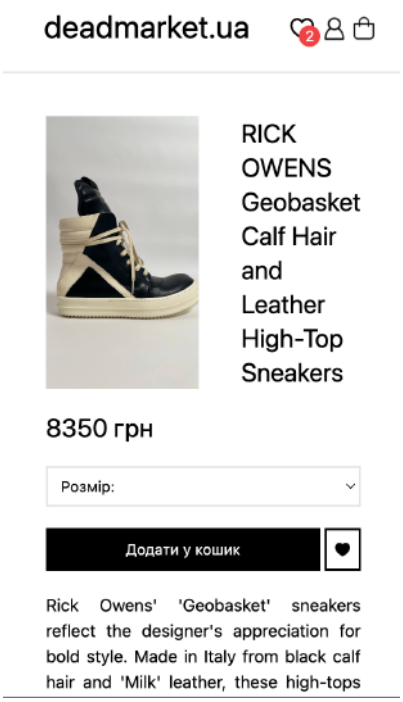

Рис. 2.16. Вигляд сторінки товару на мобільному пристрої

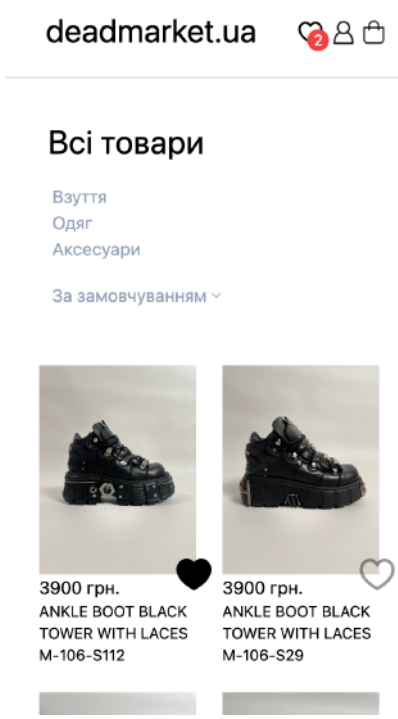

Рис. 2.17 Вигляд головної сторінки на мобільному пристрої

Для того щоб оформити замовлення треба зробити наступні кроки:

- Додати вибраний товар до кошику

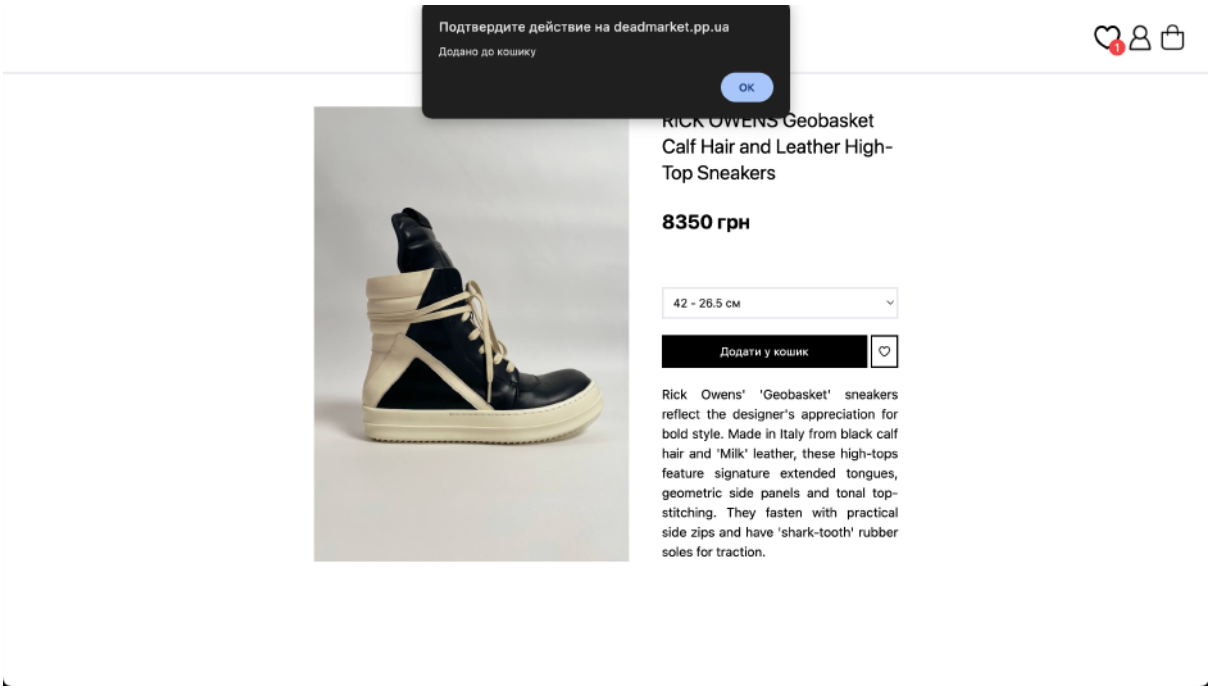

Рис. 2.18. Додавання товару в кошик

- Якщо не авторизований користувач перейде до кошику, то він побаче надпис що йому треба пройти авторизацію

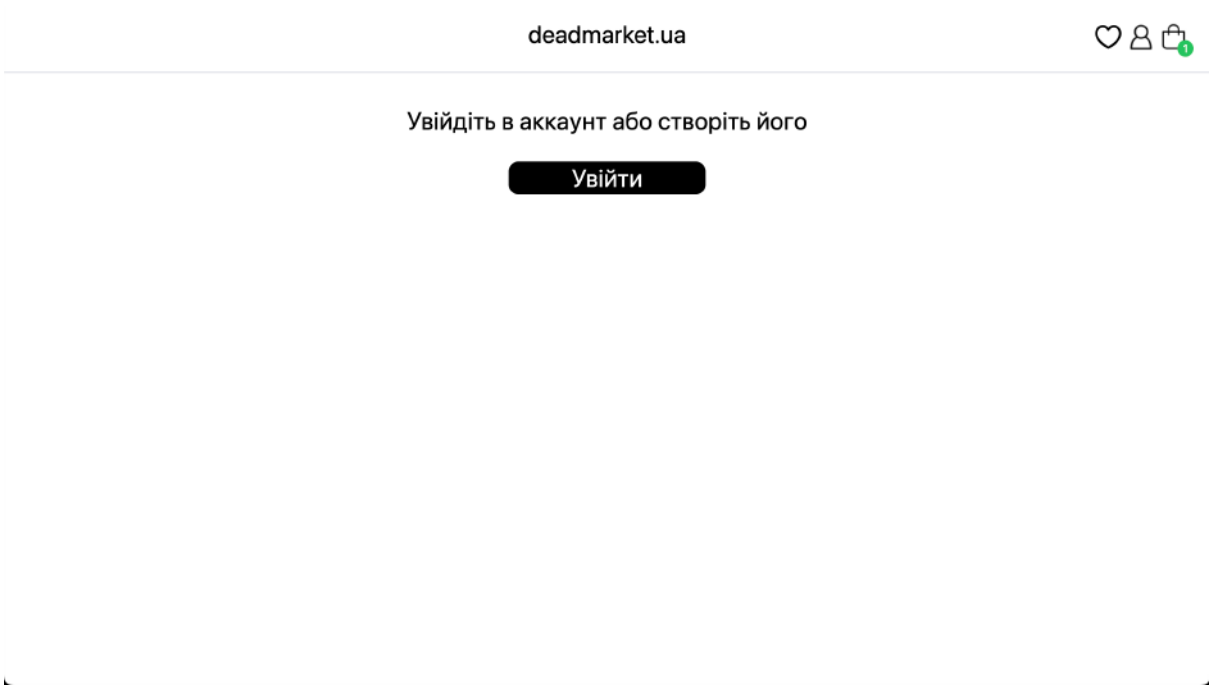

Рис. 2.19 Вигляд кошику, якщо користувач не авторизований

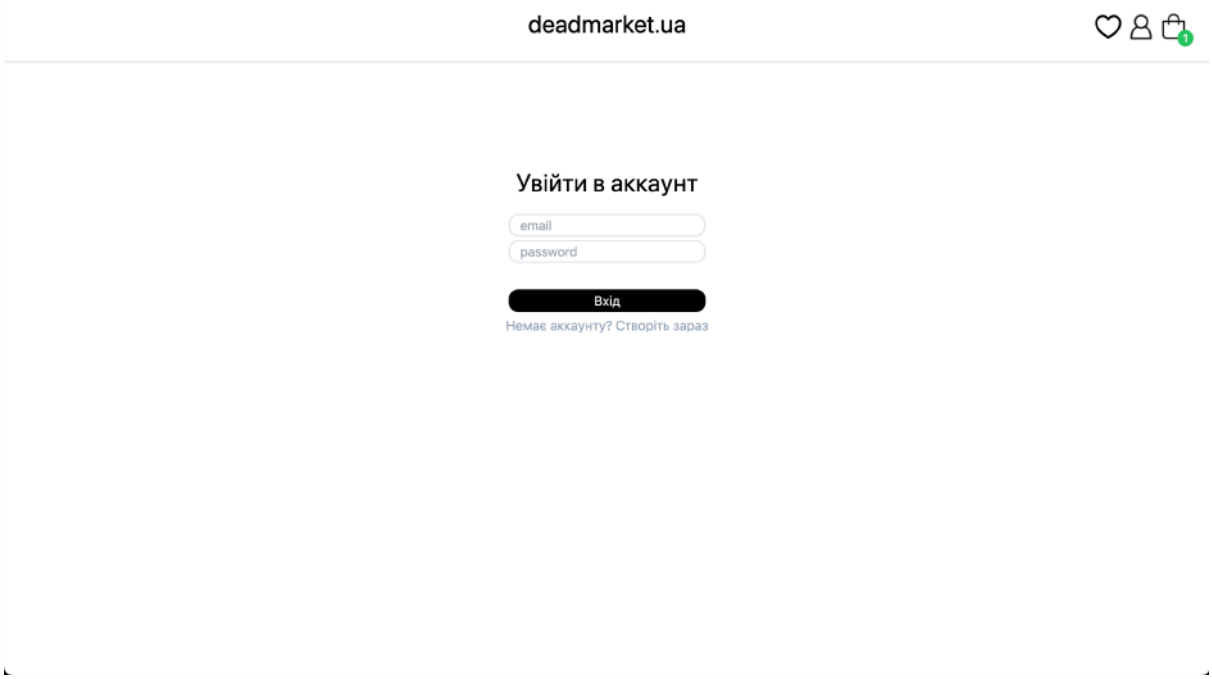

Рис. 2.20. Сторінка авторизації

- Після натиснення кнопки увійти користувач потрапляє на сторінку входу та рєєстрації. При натисненні тексту «Небає аккаунту? Створіть зараз» форма авторизації змінюється на форму реєстрації, та можно навпаки повернутись натиснувши “ Вже є аккаунт? Увійдіть в нього”

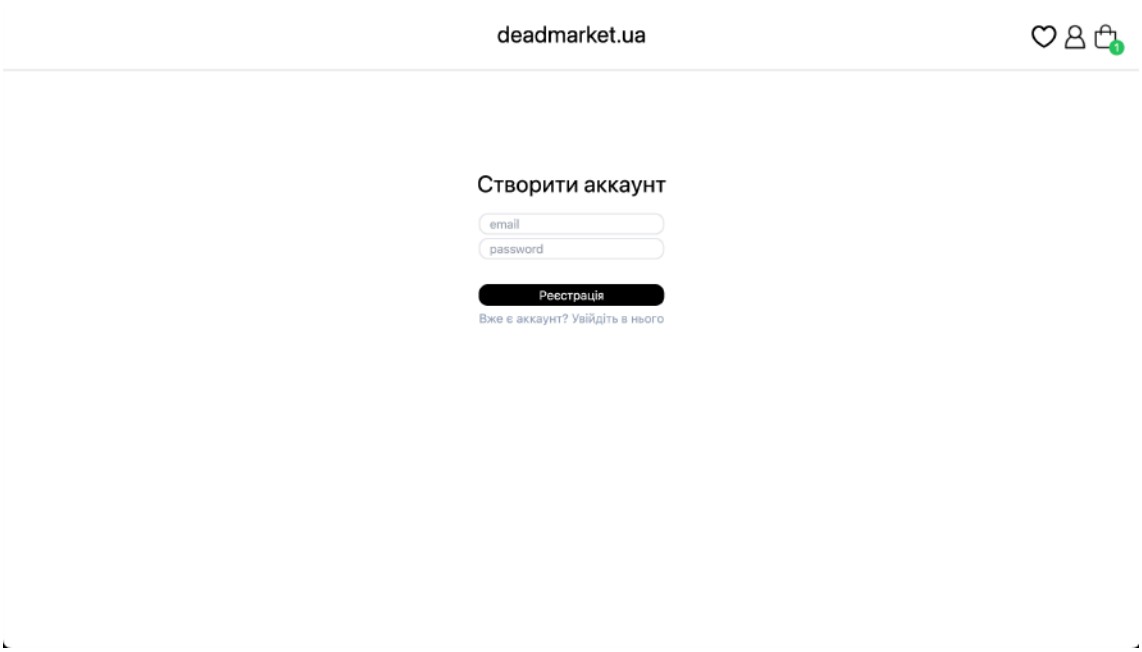

Рис. 2.21. Сторінка реєстрації

Після входу користувача він отримує доступ перейти в кошик

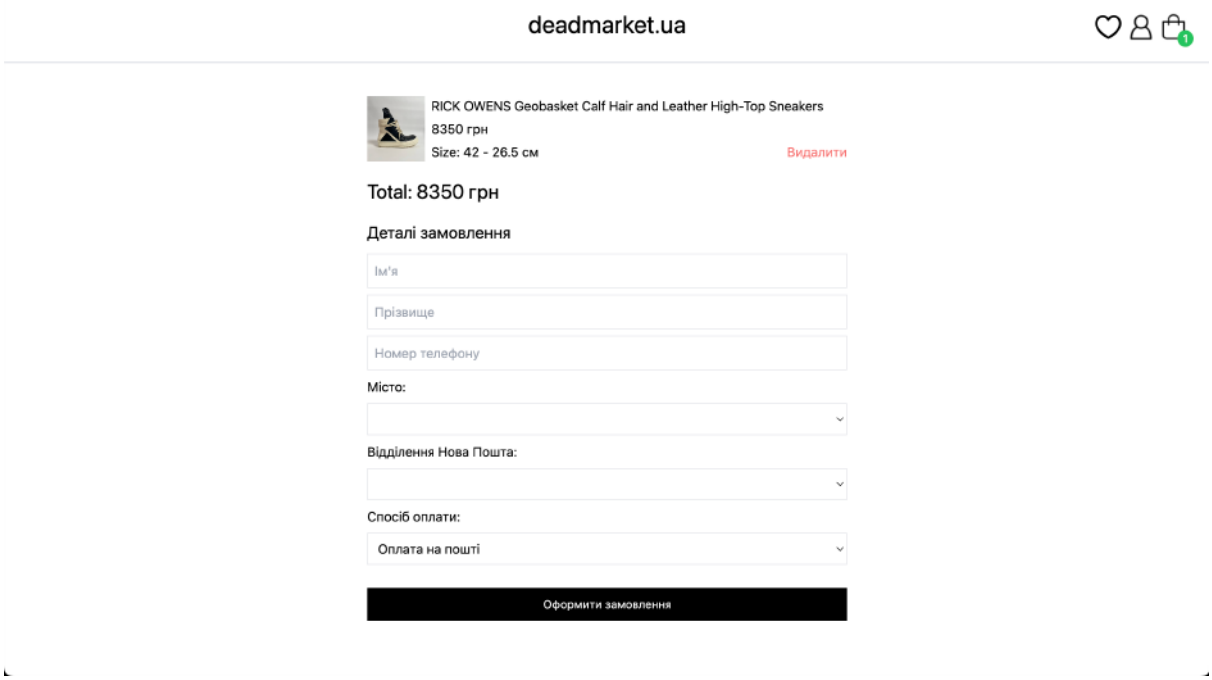

Рис. 2.22. Сторінка кошику

Якщо користувач правильно ввів всі дані, він може натиснути кнопку “Оформити замовлення”

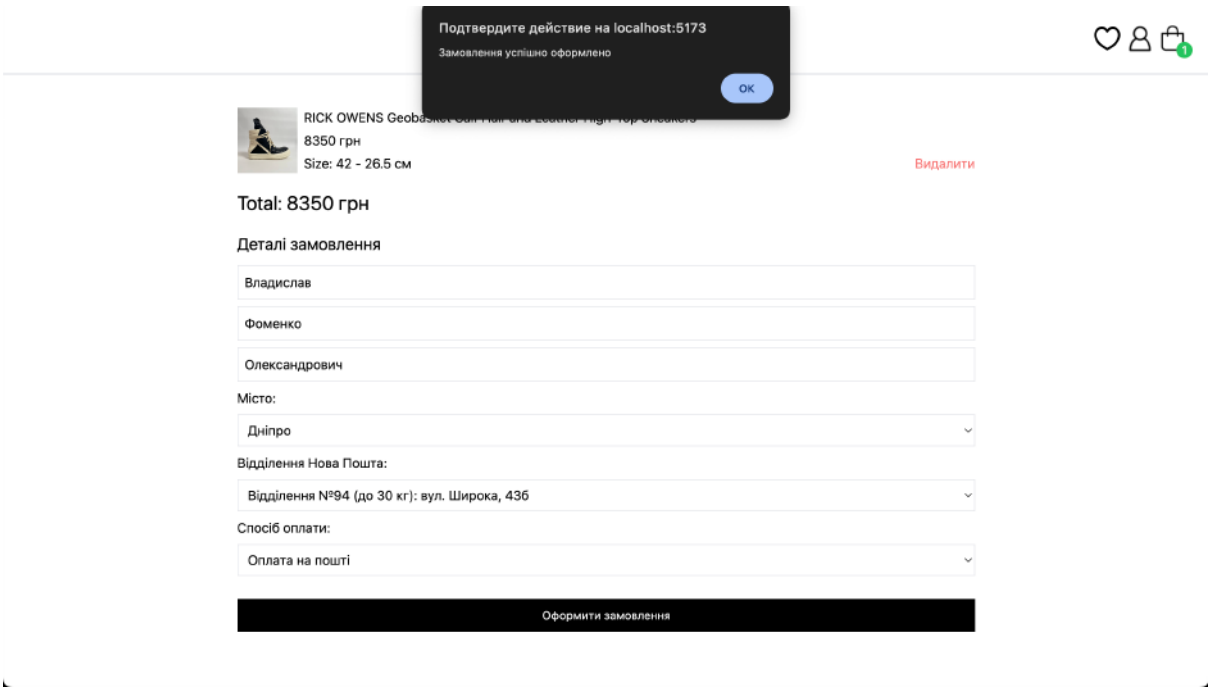

Рис. 2.23. Успішно оформенне замовлення

Потім можна перейти в профіль для перевірки коректності даних, де також можна відмінити замовлення та подивитись статус товару

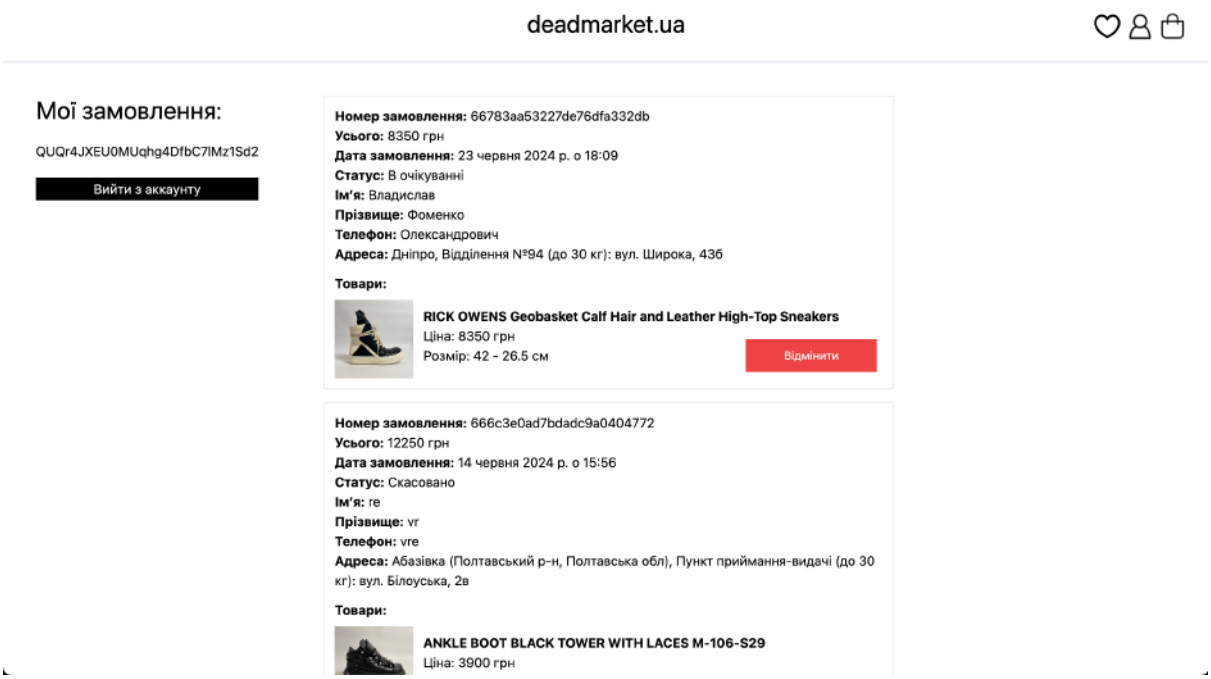

Рис. 2.24. Сторінка профілю та замовлень

Знизу сторінки додатку є розділ з інформацією, яка може знадобитись користувачу

Рис. 2.25. Футер сайту

Рис 2.26. Розділ “Про нас”

Також наявна функціональна форма зворотнього звязку. Після відправки форми повідомлення відправляється адміністраторам на пошту.

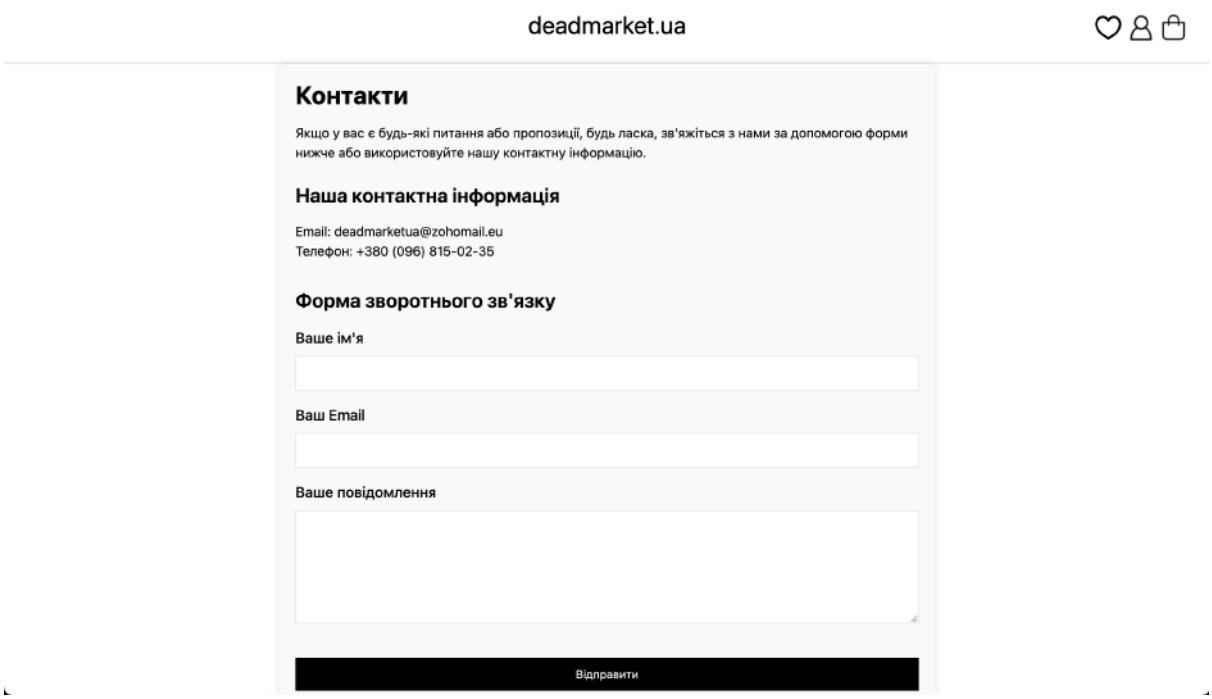

Рис. 2.27. Сторінка контактів та зворотнього зв’язку

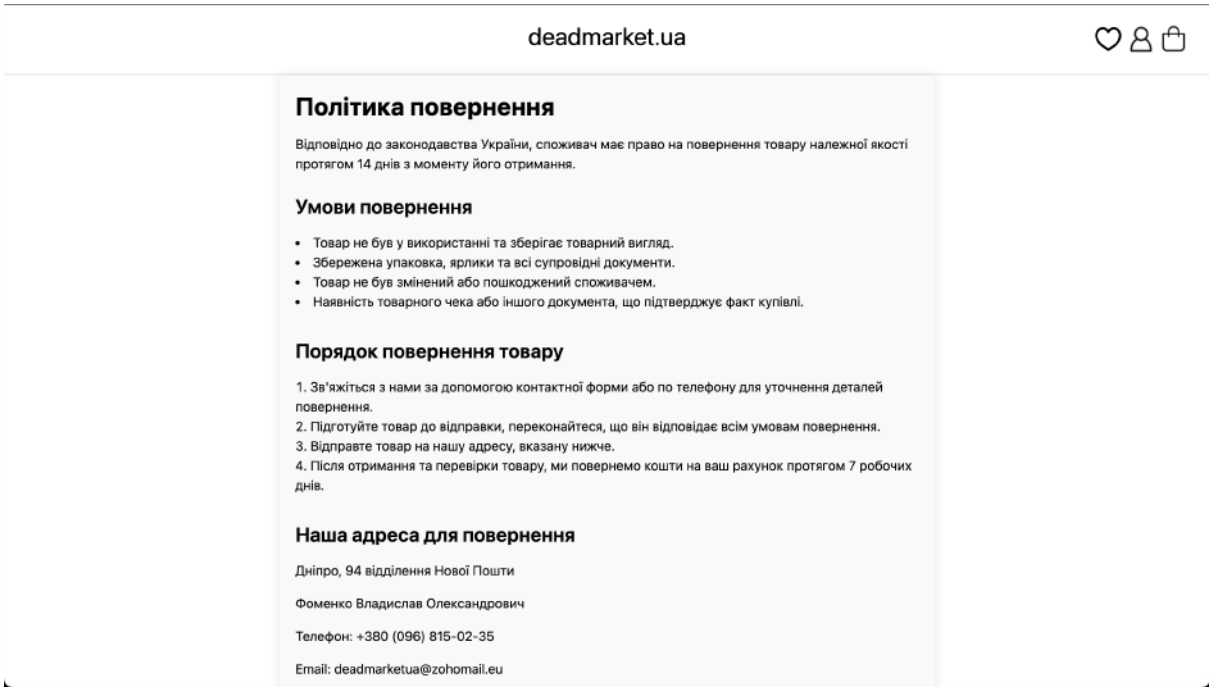

Рис. 2.28. Сторінка “Повернення товару”

На цьому основні функції для звичайного користувача закінчуються.

Для адміністратору системи створено окремі сторінки для управління товарами, та праці із замовленнями.

Після входу в спеціальний аккаунт адміністатору система нас перенаправляє на сторінку всіх замовлень, де можна видалити замовлення або змінити його статус

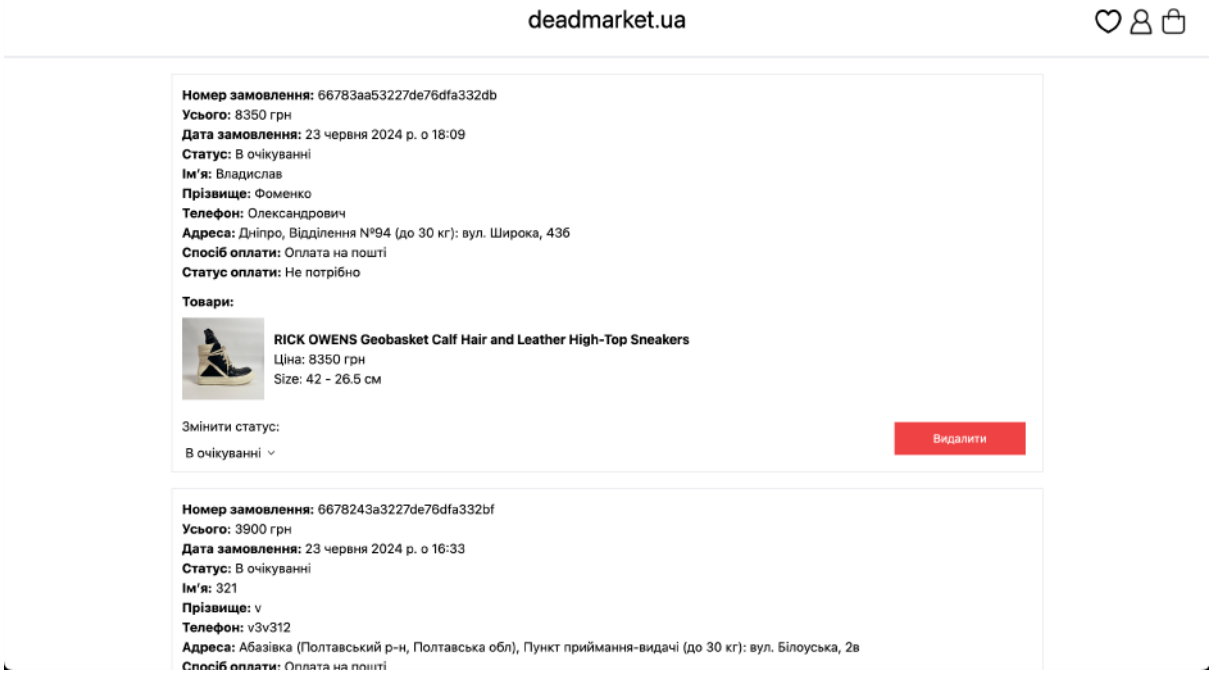

Рис. 2.29. Сторінка адміністратора по управлінню замовленнями

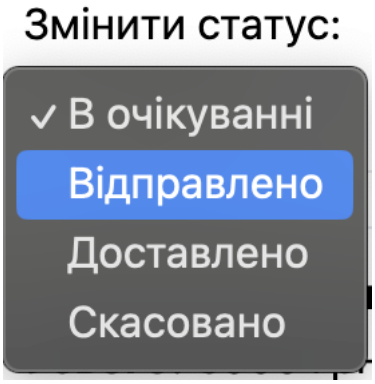

Рис. 2.30. Зміна статусу замовлення

Для переходу на сторінку додавання товарів треба ввести певну адресу. Це зроблено щоб не навантажувати інтерфейс непотрібним кодом та елементами, оскільки доступ до цієї сторінки має тільки одна людина яка ознайомлена с функціоналом додатку.

На сторінці додавання товару є форма для заповнення інформації про товар. Заповнивши всі поля форми та додавши зображення можна натиснути кнопку “Додати товар”, після чого товар буде автоматично додано в перелік товарів та він буде відображатись на сайті.

Якщо товар вже доданий, то його можна повністю відредагувати, змінивши всі поля форми і навіть зображення.

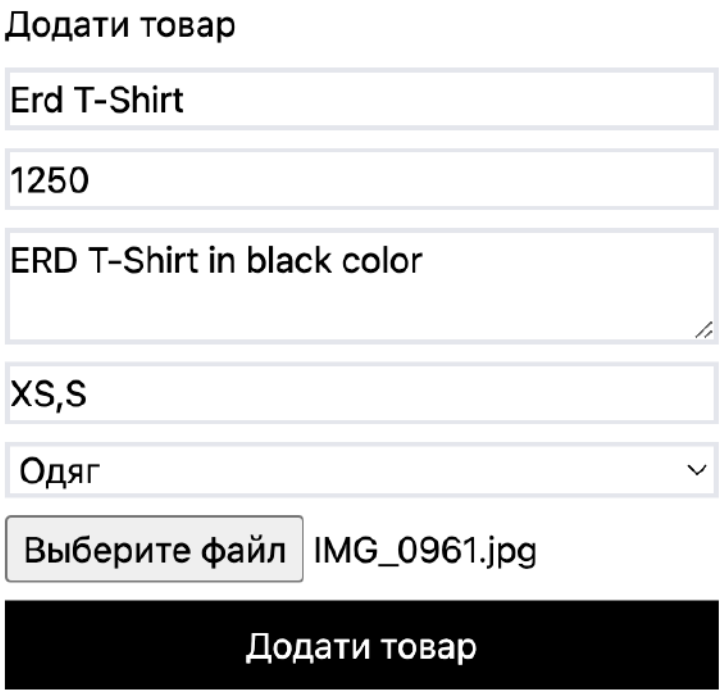

Рис. 2.31. Додавання товару

Рис. 2.32. Вибір категорії

При додаванні або редагувані товару можна вибрати одну з категорій, які відображені на рис. Доданий товар в списку виглядає наступним чином:

Рис. 2.33. Картка товару

Його можна видалити або редагувати. Після натиснення кнопки “Редагувати” форма заповнюєтся даними про товар який редагується та після внесення змін в дані його можна оновити.

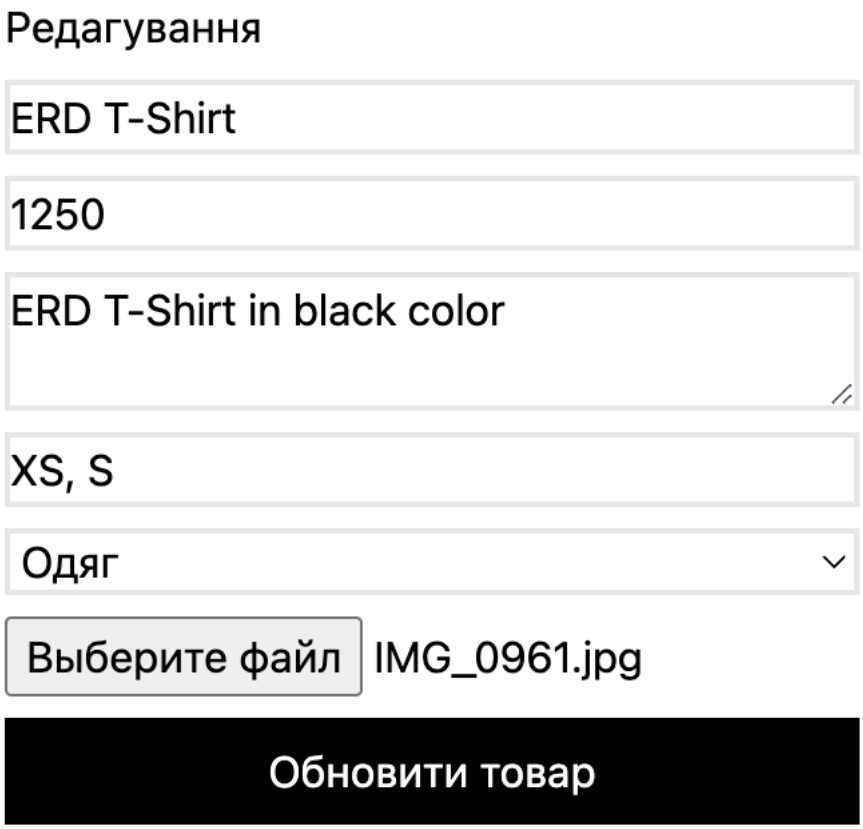

Рис. 2.34. Оновлення товару

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Вихідні дані:

1. q – передбачуване число операторів програми – 2050;
2. p – коефіцієнт корекції програми в ході її розробки – 0,1;
3. C – коефіцієнт складності програми – 1,3;
4. $C_{пр}$ – годинна заробітна плата програміста – 215 грн/год;

По статистиці “DOU” середня заробітна плата Junior JavaScript Developer становить 900$ на кінець 2023 року. На середину червня курс долару по НБУ становить 40,649 грн**.** По розрахункам в гривнях середня заробітна плата - 36584.1 грн. При робочому графіку 21 днів на місяць по 8 годин в день годинна заробітна плата буде становити близько 217.76 грн.

5. B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1.1;
6. k – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1;
7. $C_{мв}$ – вартість машино-години ЕОМ – 1,636 грн/год;
8. B_k – число виконавців – 1 людина;
9. F_p – місячний фонд робочого часу (21 день по 8 годин) – 168 годин.

При виконані квалафікаційної роботи було використано ноутбук та домашній інтернет.

Ноутбук на якому виконувалась розробка додатку споживає до 30 Вт в максимальному навантаженні. В середньому при невеликому навантажені споживає 20 Вт. Згідно з даними Мінфіну тарифи на електроенергію на червень 2024 року становить 4,32 грн/кВт*год

4,32 * 0,02 кВт = 0.086 грн/год.

Вартість тарифу інтернет зв’язку коштує 260 грн на місяць.

260 / (168 год.) = 1,55 грн/год.

Усього вартість машино-години 1.55+0.086 = 1.636

Трудомісткість розробки ПЗ можна розрахувати за допомогою формули:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин,} \qquad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p) \tag{3.2}$$

де q – передбачуване число операторів (2200),

C – коефіцієнт складності програми (1,3),

p– коефіцієнт кореляції програми в ході її розробки (0,1).

$$Q=2050 \cdot 1,3 \cdot (1+0,1)= 2931,5$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k} \text{ , людино-годин} \tag{3.3}$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (1,1);

K – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності (1);

$$t_u = (2931,5*1.1)/(85*1)= 31,35 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20\ldots25) \cdot k} \text{ , людино-годин} \tag{3.4}$$

$$t_a =2931,5/(25*1)= 117,26 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20\ldots25)\cdot k}\text{ , людино-годин} \qquad (3.5)$$

t_n = 2931,5/(25*1)=117,26 людино-годин.

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)\cdot k}\text{ , людино-годин} \qquad (3.6)$$

$t_{отл}$ = 2931,5/(5*1)=586,3 людино-годин,

– за умови комплексного налагодження завдання:

$$t_{отл}^{k} = 1{,}5\cdot t_{отл}\text{ , людино-годин} \qquad (3.7)$$

$$t_{\text{отл}}^{k} = 1{,}5\ * 586{,}3 = 879{,}45$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до}\text{ , людино-годин} \qquad (3.8)$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису

$$t_{др} = \frac{Q}{(15..20)\cdot k}\text{ , людино-годин} \qquad (3.9)$$

$t_{др}$ = 2931,5/20=146,57 людино-годин.

$t_{до}$ – трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0{,}75 \cdot t_{др}$$, людино-годин. (3.10)

$t_{до}$ = 0,75 * 146.57 = 109,92 людино-годин.

Розрахуємо витрати праці на підготовку документації за формулою (3.8):

$t_{д}$= 146,57 + 109,92 = 256,49 людино-годин.

Повертаючись до формули (3.1) зробимо розрахунки трудомісткості розробки програмного забезпечення:

t= 50 + 31,35 + 117,26 + 117,26 + 586,3 + 256,49 = 1158,66 людино-годин.

У результаті отримано, що в загальній складності необхідно 1158,66 людино-годин для розробки програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{по}$ включають витрати на заробітну плату виконавця програми $З_{зп}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн} \quad (3.11)$$

де $З_{зп}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн} \quad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$C_{ПР}$ – середня годинна заробітна плата програміста, грн/година

З урахуванням того, що середня годинна зарплата програміста становить 215 грн / год, отримуємо:

$$З_{зп} = 1158,66 \cdot 217,76 = 252309,80 \text{ грн.}$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{мв} = t_{отл} \cdot C_{мч}, \text{ грн,} \qquad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год;

$C_{мв}$ – вартість машино-години ЕОМ, грн/год.

Підставивши в формулу (3.13) відповідні значення, визначимо вартість необхідного для налагодження машинного часу:

$$З_{мв} = 586,3 \cdot 1,636 = 959,19 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{по} = 252309,80 + 959,19 = 253268,98 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$\mathbf{T} = \frac{\mathbf{t}}{\mathbf{B_k} \cdot \mathbf{F_p}}, \text{ міс,} \qquad (3.14)$$

де B_k – число виконавців (дорівнює 1);

$F_р$ – місячний фонд робочого часу (при 40 годинному робочому тижні Fp=168).

Звідси витрати на створення програмного продукту:

$$T = 1158,66 / (1*168) \approx 6,9 \text{ міс}$$

Висновок: За розрахунками трудомісткість розробленого програмного додатку становить 1158,66 людино-години, з урахуванням розробки однією людиною рівня Junior. Обчислена вартість роботи над створенням програми становить 253268,98 гривень. Приблизний очікуваний період створення програмного забезпечення із графіком 21 робочій день на місяць та 8 годин в день складає 6,9 місяців.

ВИСНОВКИ

В рамках кваліфікаційної роботи був розроблений повністю функціональний веб-додаток який покриває всі потреби для інтернет магазину, що в подальшому покращить ефективність його роботи. Основна мета розробки була спрямована на реалізацію автоматичного прийому замовлень від користувачів додатку.

Актуальність проблеми пов’язана з тим що ринок електронної комерції поступово збільшуються, оскільки, в умовах війни, покупцям та продавцям зручніше користуватись дистанційними засобами.

Додаток був розроблений за допомогою таких сучасних і популярних технологій, як Vue.js та Express.js. Вони мають довгострокову підтримку та можливість подальшого масштабування. Основна логіка системи побудована на посиланні запитів із клієнтської частини на сервер. Додаток також використовує такі допоміжні сервіси як MongoDB Atlas для зберігання даних, та Firebase AUTH для реалізації реєстрації.

У процесі виконання кваліфікаційної роботи була визначена кількість людських та матеріальних ресурсів, яких потребує система для її розробки. Згідно розрахунків, витрати на створення програмного продукту становлять 253268,98 грн. Час, який необхідних для розробки інтернет-магазину, склав 1158,66 людино-годин, що приблизно відповідає 6,9 місяцям роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація Firebase. URL: https://firebase.google.com/docs?hl=ru (дата звернення: 10.05.2024).

2. "Концентрат малого бізнесу" - стаття про сучасну електрону комерцію. URL: https://forbes.ua/innovations/kontsentrat-malogo-biznesu-marketpleys-promua-zachepiv-dno-povernuvsya-do-zrostannya-i-dae-klientiv-dlya-40-000-pidpriemtsiv-intervyu-mikoli-palienka-11012023-10984 (дата звернення: 15.05.2024).

3. Офіційна документація Express.js. URL: https://expressjs.com/en/guide/routing.html (дата звернення: 03.05.2024).

4. Офіційна документація Pinia. URL: https://pinia.vuejs.org/introduction.html (дата звернення: 02.05.2024).

5. Офіційна документація Vue.js. URL: https://vuejs.org/guide/introduction.html (дата звернення: 25.04.2024).

6. "Роздрібна торгівля проти електронної комерції" - стаття про відмінності онлайн комерції та фізичних магазинів. URL: https://www.doola.com/uk/blog/retail-vs-ecommerce/ (дата звернення: 15.05.2024).

7. Середня заробітна плата Junior JavaScript Developer за версією DOU.ua. URL: https://jobs.dou.ua/salaries/?period=2023-12&position=Junior%20SE&technology=JavaScript.

8. Model-View-Controller - опис концепції. URL: https://ru.wikipedia.org/wiki/Model-View-Controller (дата звернення: 20.05.2024).

9. Вебінтерфейс - Wikipedia. URL: https://uk.wikipedia.org/wiki/Вебінтерфейс (дата звернення: 17.05.2024).

10. Документація MongoDB. URL: https://www.mongodb.com/docs/ (дата звернення: 04.05.2024).

11. Документація Mongoose. URL: https://mongoosejs.com/docs/documents.html (дата звернення: 01.05.2024).

12. Документація Postman. URL: https://learning.postman.com/docs/introduction/overview/ (дата звернення: 10.05.2024).

13. Метод HTTP-запитів. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods (дата звернення: 24.05.2024).

14. Офіційна документація "Tailwind". URL: https://tailwindcss.com/docs/ (дата звернення: 02.05.2024).

15. Учасники проектів Вікімедіа. Front end та back end - Wikipedia. URL: https://uk.wikipedia.org/wiki/Front_end_та_back_end (дата звернення: 15.05.2024).

16. Client-Server Overview - MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview (дата звернення: 09.05.2024).

17. HTTP. URL: https://ru.wikipedia.org/wiki/HTTP (дата звернення: 11.05.2024).

18. MDN Web Docs - Javascript документація. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference (дата звернення: 28.04.2024).

19. REST API - Wikipedia. URL: https://ru.wikipedia.org/wiki/REST (дата звернення: 25.05.2024).
20. VPS-сервер. URL: https://server.ua/ua/vps (дата звернення: 30.05.2024).

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

```
//index.js
import mongoose from "mongoose";
import express from "express";
import multer from "multer";
import path from "path";
import cors from "cors";
import axios from "axios";
import admin from "firebase-admin";
import crypto from "crypto";
import nodemailer from "nodemailer";
import { fileURLToPath } from "url";
import bodyParser from "body-parser"; // Add this import

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

const port = 5000;
const mongoUri =
  "mongodb+srv://vladfomen:pass1234@cluster0.ljicrxs.mongodb.net/";

mongoose
  .connect(mongoUri, { useNewUrlParser: true, useUnifiedTopology: true })
  .then(() => console.log("Connected to MongoDB"))
  .catch((err) => console.error("Could not connect to MongoDB", err));

const app = express();

const allowedOrigins = ["http://localhost:5173", "https://deadmarket.pp.ua"];
const corsOptions = {
  origin: function (origin, callback) {
    if (allowedOrigins.includes(origin) || !origin) {
      callback(null, true);
    } else {
      callback(new Error("Not allowed by CORS"));
    }
  },
};

app.use(cors(corsOptions));
app.use(express.json({ limit: "50mb" }));
app.use(express.urlencoded({ limit: "50mb", extended: true }));

app.use(express.static(path.join(__dirname, "public")));

// Middleware for parsing raw body
app.use(bodyParser.raw({ type: "application/json" }));
const transporter = nodemailer.createTransport({
```

```
  host: "smtp.zoho.eu",
  port: 465,
  secure: true,
  auth: {
    user: "deadmarketua@zohomail.eu", // Замените на вашу почту Zoho
    pass: "ParisCity", // Замените на пароль
  },
});

app.post("/send-email", (req, res) => {
  const { name, email, message } = req.body;

  if (!name || !email || !message) {
    return res.status(400).send("All fields are required");
  }

  const mailOptions = {
    from: "deadmarketua@zohomail.eu", // Замените на вашу почту Zoho
    to: "deadmarketua@zohomail.eu", // Замените на вашу почту Zoho
    subject: `Contact form submission from ${name}`,
    text: `You received a new message from ${name} (${email}):\n\n${message}`,
  };

  transporter.sendMail(mailOptions, (error, info) => {
    if (error) {
      return res.status(500).send(error.toString());
    }
    res.status(200).send("Email sent: " + info.response);
  });
});

// Middleware for CSP headers and nonce
// Middleware for CSP headers and nonce
app.use((req, res, next) => {
  res.locals.nonce = crypto.randomBytes(16).toString("hex");
  res.setHeader(
    "Content-Security-Policy",
    `default-src 'self'; script-src 'self' 'nonce-${res.locals.nonce}' https://js.stripe.com
https://apis.google.com; frame-src https://js.stripe.com; connect-src 'self' https://api.stripe.com
https://identitytoolkit.googleapis.com; img-src 'self' data: https://*.stripe.com; style-src 'self'
'unsafe-inline' https://fonts.googleapis.com; font-src 'self' https://fonts.gstatic.com;`
  );
  next();
});

admin.initializeApp({
  credential: admin.credential.cert(
    "./jwt-firebase-vue3-31ba4-936daffcd877.json"
  ),
});

const productSchema = new mongoose.Schema({
```

```
  name: String,
  price: Number,
  description: String,
  sizes: [String],
  category: String,
  image: String,
});
const orderSchema = new mongoose.Schema({
  products: [
    {
      productId: { type: mongoose.Schema.Types.ObjectId, ref: "Product" },
      name: String,
      price: Number,
      image: String,
      selectedSize: String,
    },
  ],
  total: Number,
  userId: String,
  status: { type: String, default: "Pending" },
  customerName: String,
  customerSurname: String,
  customerPhone: String,
  customerAddress: String,
  createdAt: { type: Date, default: Date.now },
  paymentMethod: String,
  paymentStatus: String,
});

const Product = mongoose.model("Product", productSchema);
const Order = mongoose.model("Order", orderSchema);

const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, "uploads/");
  },
  filename: function (req, file, cb) {
    cb(null, Date.now() + path.extname(file.originalname));
  },
});

const upload = multer({ storage: storage });

const checkAdmin = async (req, res, next) => {
  const authorizationHeader = req.headers.authorization;
  if (!authorizationHeader) {
    return res.status(403).send("Forbidden: No authorization header provided");
  }

  const idToken = authorizationHeader.split("Bearer ")[1];
  if (!idToken) {
    return res.status(403).send("Forbidden: Invalid authorization format");
```

```
  }

  try {
    const decodedToken = await admin.auth().verifyIdToken(idToken);
    if (decodedToken.email === "vladfomen@gmail.com") {
      next();
    } else {
      res.status(403).send("Forbidden: You are not an admin");
    }
  } catch (error) {
    console.error("Error verifying token:", error);
    res.status(403).send("Forbidden: Invalid authorization token");
  }
};

app.post(
  "/admin/products",
  checkAdmin,
  upload.single("image"),
  async (req, res) => {
    try {
      const sizes = JSON.parse(req.body.sizes);
      const newProduct = new Product({
        name: req.body.name,
        price: req.body.price,
        description: req.body.description,
        sizes: sizes,
        category: req.body.category,
        image: req.file.filename,
      });

      await newProduct.save();
      res.status(201).send(newProduct);
    } catch (error) {
      res.status(400).send(error);
    }
  }
);

app.get("/products", async (req, res) => {
  try {
    const products = await Product.find();
    res.status(200).send(products);
  } catch (error) {
    res.status(500).send(error);
  }
});

app.delete("/admin/products/:id", checkAdmin, async (req, res) => {
  try {
    const productId = req.params.id;
    const deletedProduct = await Product.findByIdAndDelete(productId);
```

```
    if (!deletedProduct) {
      return res.status(404).send({ message: "Product not found" });
    }
    res.status(200).send({ message: "Product deleted successfully" });
  } catch (error) {
    res.status(500).send(error);
  }
});

app.put(
  "/admin/products/:id",
  checkAdmin,
  upload.single("image"),
  async (req, res) => {
    try {
      const productId = req.params.id;
      const sizes = JSON.parse(req.body.sizes);

      const updateData = {
        name: req.body.name,
        price: req.body.price,
        description: req.body.description,
        sizes: sizes,
        category: req.body.category,
      };

      if (req.file) {
        updateData.image = req.file.filename;
      }

      const updatedProduct = await Product.findByIdAndUpdate(
        productId,
        updateData,
        { new: true }
      );
      if (!updatedProduct) {
        return res.status(404).send({ message: "Product not found" });
      }
      res.status(200).send(updatedProduct);
    } catch (error) {
      res.status(400).send(error);
    }
  }
);

app.post("/orders", async (req, res) => {
  try {
    console.log("Received order data:", req.body);

    const newOrder = new Order({
      products: req.body.products,
      total: req.body.total,
```

```
      userId: req.body.userId,
      customerName: req.body.customerName,
      customerSurname: req.body.customerSurname,
      customerPhone: req.body.customerPhone,
      customerAddress: req.body.customerAddress,
      paymentMethod: req.body.paymentMethod,
      paymentStatus:
        req.body.paymentMethod === "online-payment"
          ? "Pending"
          : "Not Required",
    });

    await newOrder.save();
    res.status(201).send(newOrder);
  } catch (error) {
    console.error("Error saving order:", error);
    res.status(400).send(error);
  }
});

app.get("/orders", checkAdmin, async (req, res) => {
  try {
    const orders = await Order.find().populate("products.productId");
    res.status(200).send(orders);
  } catch (error) {
    console.error(error);
    res.status(500).send(error);
  }
});

app.get("/orders/:userId", async (req, res) => {
  try {
    const userId = req.params.userId;
    console.log(`Fetching orders for user: ${userId}`);
    const orders = await Order.find({ userId }).populate("products.productId");
    console.log(`Orders found: ${orders.length}`);
    res.status(200).send(orders);
  } catch (error) {
    console.error(error);
    res.status(500).send(error);
  }
});

app.put("/orders/:id/status", checkAdmin, async (req, res) => {
  try {
    const orderId = req.params.id;
    const { status } = req.body;
    const updatedOrder = await Order.findByIdAndUpdate(
      orderId,
      { status },
      { new: true }
    );
```

```
    if (!updatedOrder) {
      return res.status(404).send({ message: "Order not found" });
    }
    res.status(200).send(updatedOrder);
  } catch (error) {
    res.status(400).send(error);
  }
});

app.delete("/admin/orders/:id", checkAdmin, async (req, res) => {
  try {
    const orderId = req.params.id;
    const deletedOrder = await Order.findByIdAndDelete(orderId);
    if (!deletedOrder) {
      return res.status(404).send({ message: "Order not found" });
    }
    res.status(200).send({ message: "Order deleted successfully" });
  } catch (error) {
    res.status(500).send(error);
  }
});

app.put("/orders/:id/cancel", async (req, res) => {
  try {
    const orderId = req.params.id;
    const updatedOrder = await Order.findByIdAndUpdate(
      orderId,
      { status: "Cancelled" },
      { new: true }
    );
    if (!updatedOrder) {
      return res.status(404).send({ message: "Order not found" });
    }
    res.status(200).send(updatedOrder);
  } catch (error) {
    res.status(400).send(error);
  }
});

app.get("/novaposhta/cities", async (req, res) => {
  const apiKey = "d9996df2f1c38a49e20d70b4c815682a";
  try {
    console.log("Fetching cities");
    const response = await axios.post("https://api.novaposhta.ua/v2.0/json/", {
      apiKey: apiKey,
      modelName: "Address",
      calledMethod: "getCities",
      methodProperties: {},
    });
    console.log("Cities response:", response.data);
    res.status(200).send(response.data.data);
  } catch (error) {
```

```
    console.error(
      "Error fetching cities:",
      error.response ? error.response.data : error.message
    );
    res.status(500).send(error);
  }
});

app.get("/novaposhta/branches", async (req, res) => {
  const { cityRef } = req.query;
  const apiKey = "d9996df2f1c38a49e20d70b4c815682a";
  try {
    console.log(`Fetching branches for cityRef: ${cityRef}`);
    const response = await axios.post("https://api.novaposhta.ua/v2.0/json/", {
      apiKey: apiKey,
      modelName: "AddressGeneral",
      calledMethod: "getWarehouses",
      methodProperties: {
        CityRef: cityRef,
      },
    });
    console.log("Branches response:", response.data);
    res.status(200).send(response.data.data);
  } catch (error) {
    console.error(
      "Error fetching branches:",
      error.response ? error.response.data : error.message
    );
    res.status(500).send(error);
  }
});

app.use("/uploads", express.static("uploads"));

app.get("*", (req, res) => {
  res.sendFile(path.join(__dirname, "public", "index.html"));
});

app.listen(port, () => {
  console.log(`Server listening at port ${port}`);
                                                    });
//App.vue
<template>
  <div class="min-h-screen">
    <MainHeader :favoriteOpenClick="FavoriteOpenClick"
:removeFromFavorite="removeFromFavorite" />
    <router-view :shoes="shoes"></router-view>
  </div>
  <MainFooter></MainFooter>
</template>

<script setup>
```

```
import MainHeader from './components/MainHeader.vue'
import MainFooter from './components/MainFooter.vue'
import { ref, onMounted } from 'vue'
import { getAuth, onAuthStateChanged } from 'firebase/auth'
import { useUserStore } from '@/stores/user'
import { provide } from 'vue'
import axios from 'axios'

const userStore = useUserStore()
const shoes = ref([])

const getShoes = async () => {
  try {
    const response = await axios.get('https://deadmarket.pp.ua/products')
    shoes.value = response.data
  } catch (error) {
    console.error('Error fetching products:', error)
  }
}

getShoes()

provide('shoes', shoes)
onMounted(() => {
  onAuthStateChanged(getAuth(), (user) => {
    if (user) {
      userStore.userId = user.uid
    } else {
      userStore.userId = ''
    }
  })
})
</script>

<style scoped></style>
//HomaPage.vue
<template>
  <main>
    <div class="sm:flex z-0">
      <NavigationMenu @filter-category="filterCategory" @sort-items="sortItems" />
      <div class="m-auto">
        <div class="flex-col mt-10 m-5">
          <div class="grid lg:grid-cols-4 sm:grid-cols-3 grid-cols-2 w-fit">
            <ItemCard
              @click="navigateToDetail(shoe._id)"
              v-for="shoe in filteredShoes"
              :key="shoe._id"
              class="m-3 cursor-pointer hover:drop-shadow-2xl"
              :shoe="shoe"
            />
            <div></div>
          </div>
```

```
      </div>
    </div>
  </div>
 </main>
</template>

<script setup>
import { ref, computed } from 'vue'
import { useRouter } from 'vue-router'

import ItemCard from '@/components/ItemCard.vue'
import NavigationMenu from '@/components/NavigationMenu.vue'

const router = useRouter()

const props = defineProps({
 shoes: {
  type: Array,
  required: true
 }
})

const selectedCategory = ref(null)
const sortOrder = ref('default')

const filterCategory = (category) => {
 selectedCategory.value = category
}

const sortItems = (order) => {
 sortOrder.value = order
}

const filteredShoes = computed(() => {
 let filtered = props.shoes

 if (selectedCategory.value) {
  filtered = filtered.filter((shoe) => shoe.category === selectedCategory.value)
 }

 if (sortOrder.value === 'cheapest') {
  filtered = filtered.sort((a, b) => a.price - b.price)
 } else if (sortOrder.value === 'expensive') {
  filtered = filtered.sort((a, b) => b.price - a.price)
 } else if (sortOrder.value === 'newest') {
  filtered = filtered.sort((a, b) => new Date(b.createdAt) - new Date(a.createdAt))
 }

 return filtered
})

const navigateToDetail = (id) => {
```

```
  router.push({ name: 'product-detail', params: { id } })
}
</script>

<style scoped>
/* Add your styles here */
</style>
//router.js
import { createRouter, createWebHistory } from 'vue-router'
import HomePage from '../pages/HomePage.vue'
import ProductDetail from '../pages/ProductDetail.vue'
import CartPage from '../pages/CartPage.vue'
import LoginProfile from '../pages/LoginProfile.vue'
import AdminEdit from '../pages/AdminEdit.vue'
import ProfilePage from '../pages/ProfilePage.vue'
import AdminOrders from '../pages/AdminOrders.vue' // Новый компонент
import AboutUs from '../pages/AbousUs.vue'
import ContactUs from '../pages/ContactUs.vue'
import ReturnPage from '../pages/ReturnPage.vue'
import { getAuth, onAuthStateChanged } from 'firebase/auth'
const checkAuth = (to, from, next) => {
  let isAuth = false
  onAuthStateChanged(getAuth(), (user) => {
    if (user && !isAuth) {
      isAuth = true
      next()
    } else if (!user && !isAuth) {
      isAuth = true
      next('/login')
    }
  })
}

const checkAdmin = (to, from, next) => {
  const auth = getAuth()
  onAuthStateChanged(auth, (user) => {
    if (user && user.email === 'vladfomen@gmail.com') {
      next()
    } else {
      next('/login')
    }
  })
}

const routes = [
  { path: '/', name: 'home', component: HomePage },
  { path: '/product/:id', name: 'product-detail', component: ProductDetail, props: true },
  { path: '/cart', name: 'cart', component: CartPage },
  { path: '/login', name: 'login', component: LoginProfile },
  { path: '/profile', name: 'profile', component: ProfilePage, beforeEnter: checkAuth },
  { path: '/admin/edit', name: 'admin-edit', component: AdminEdit, beforeEnter: checkAdmin }, //
Только админ
```

```
  { path: '/admin/orders', name: 'admin-orders', component: AdminOrders, beforeEnter:
checkAdmin }, // Только админ
  { path: '/aboutus', name: 'about-us', component: AboutUs },
  { path: '/contactus', name: 'contact-us', component: ContactUs },
  { path: '/return', name: 'return', component: ReturnPage }
]

const router = createRouter({
  history: createWebHistory(),
  routes
})

export default router
//CartPage.vue
<template>
  <div v-if="isLogin" class="flex flex-col">
    <h1 v-if="!cart.length" class="text-3xl mt-10 text-center">В вашому кошику немає
товарів</h1>
    <div class="sm:m-auto max-sm:m-3" v-if="cart.length > 0">
      <ul class="sm:mt-10">
        <li v-for="product in cart" :key="product._id" class="mt-2">
          <div class="flex">
            <img
              :src="`https://deadmarket.pp.ua/uploads/${product.image}`"
              alt="Product Image"
              class="w-20 h-20 object-cover"
            />
            <div class="flex justify-between w-full pl-2 flex-col relative">
              <p>{{ product.name }}</p>
              <p>{{ product.price }} грн</p>
              <p>Size: {{ product.selectedSize }}</p>
              <button
                @click="removeFromCart(product._id)"
                class="text-red-400 hover:text-red-500 transition absolute right-0 bottom-0"
              >
                Видалити
              </button>
            </div>
          </div>
        </li>
      </ul>
      <h2 class="mt-5 text-2xl">Total: {{ cartTotal }} грн</h2>

      <div class="mt-5">
        <h3 class="text-xl mb-3">Деталі замовлення</h3>
        <form @submit.prevent="placeOrder">
          <input
            v-model="customerName"
            type="text"
            placeholder="Ім'я"
            class="border p-2 mb-2 w-full"
            required
```

```
/>
<input
  v-model="customerSurname"
  type="text"
  placeholder="Прізвище"
  class="border p-2 mb-2 w-full"
  required
/>
<input
  v-model="customerPhone"
  type="text"
  placeholder="Номер телефону"
  class="border p-2 mb-2 w-full"
  required
/>

<label for="city" class="block mb-2">Місто:</label>
<select
  v-model="selectedCity"
  id="city"
  class="border p-2 mb-2 w-full"
  @change="fetchBranches"
  required
>
  <option v-for="city in cities" :key="city.Ref" :value="city">
    {{ city.Description }}
  </option>
</select>

<label for="branch" class="block mb-2">Відділення Нова Пошта:</label>
<select v-model="selectedBranch" id="branch" class="border p-2 mb-2 w-full" required>
  <option v-for="branch in branches" :key="branch.Ref" :value="branch">
    {{ branch.Description }}
  </option>
</select>

<label for="payment" class="block mb-2">Спосіб оплати:</label>
<select
  v-model="selectedPaymentMethod"
  id="payment"
  class="border p-2 mb-2 w-full"
  required
>
  <option value="cash-on-delivery">Оплата на пошті</option>
  <option disabled value="online-payment">Оплатити онлайн</option>
</select>

<button
  type="submit"
  class="mt-5 border-2 text-white bg-black border-black p-2 w-full text-sm"
>
  Оформити замовлення
```

```
          </button>
        </form>
      </div>
    </div>
  </div>
  <div v-else class="text-3xl mt-10 text-center">
    <p>Увійдіть в аккаунт або створіть його</p>
    <router-link to="/login">
      <button
        :disabled="isLoading"
        class="border-2 w-60 rounded-xl mt-8 bg-black text-white border-black active:bg-white
active:text-black transition mb-1"
        type="submit"
      >
        Увійти
      </button></router-link
    >
  </div>
</template>

<script setup>
import { ref, computed, onMounted } from 'vue'
import { useCartStore } from '@/stores/cart'
import { getAuth } from 'firebase/auth'
import axios from 'axios'
import { useRouter } from 'vue-router'
import { useUserStore } from '@/stores/user'

const router = useRouter()

const userStore = useUserStore()
const cartStore = useCartStore()
const cart = ref(cartStore.cart)
const isLogin = ref(userStore.userId)

const customerName = ref('')
const customerSurname = ref('')
const customerPhone = ref('')
const selectedCity = ref('')
const selectedBranch = ref('')
const selectedPaymentMethod = ref('cash-on-delivery') // Новый выбор способа оплаты
const cities = ref([]) // Список городов
const branches = ref([]) // Список отделений

const removeFromCart = (productId) => {
  cartStore.removeProductFromCart(productId)
  cart.value = cartStore.cart // Обновление состояния корзины
}

const cartTotal = computed(() => cartStore.cartTotal)

const fetchCities = async () => {
```

```
  try {
    const response = await axios.get('https://deadmarket.pp.ua/novaposhta/cities')
    cities.value = response.data
  } catch (error) {
    console.error('Error fetching cities:', error)
    alert('Error fetching cities')
  }
}

const fetchBranches = async () => {
  try {
    const response = await axios.get('https://deadmarket.pp.ua/novaposhta/branches', {
      params: { cityRef: selectedCity.value.Ref }
    })
    branches.value = response.data
  } catch (error) {
    console.error('Error fetching branches:', error)
    alert('Error fetching branches')
  }
}

const placeOrder = async () => {
  try {
    const auth = getAuth()
    const user = auth.currentUser
    if (!user) {
      alert('Будь ласка увійдіть в аккаунт або створіть його')
      router.push({ name: 'login' })
      return
    }

    const orderProducts = cart.value.map((product) => ({
      productId: product._id,
      name: product.name,
      price: product.price,
      image: product.image,
      selectedSize: product.selectedSize // Добавляем выбранный размер
    }))

    const orderData = {
      products: orderProducts,
      total: cartTotal.value,
      userId: user.uid,
      customerName: customerName.value,
      customerSurname: customerSurname.value,
      customerPhone: customerPhone.value,
      customerAddress: `${selectedCity.value.Description},
${selectedBranch.value.Description}`,
      paymentMethod: selectedPaymentMethod.value // Добавляем способ оплаты
    }

    await axios.post('https://deadmarket.pp.ua/orders', orderData)
```

```
    // Обработка оплаты на пошті або онлайн
    if (selectedPaymentMethod.value === 'cash-on-delivery') {
      alert('Замовлення успішно оформлено')
      cartStore.clearCart()
      cart.value = cartStore.cart // Обновление состояния корзины
    } else {
      alert('Оплата онлайн зараз недоступна')
    }
  } catch (error) {
    console.error('Error placing order:', error)
    alert('Error placing order')
  }
}

onMounted(() => {
  fetchCities()
})
</script>

<style scoped></style>

//MainHeader.vue
<template>
  <div class="text-center border-b-2 pb-5 pt-5">
    <div class="flex justify-end">
      <h1 class="text-3xl sm:absolute items-center justify-center w-full pb-1">
        <router-link to="/"> deadmarket.ua </router-link>
      </h1>
      <div class="relative flex justify-end mr-5 items-center">
        <a class="mr-2 relative" href="#" @click="favoriteStore.FavoriteOpenClick">
          <div
            v-if="favoriteStore.itemsInFavorite.length > 0"
            class="rounded-full absolute bg-red-500 w-5 h-5 text-white text-sm -right-1.5 -bottom-
1.5"
          >
            {{ favoriteStore.itemsInFavorite.length }}
          </div>
          <img class="w-8" src="/favorite.png" />
        </a>
        <router-link to="/profile">
          <img class="w-8" src="/profile.png" />
        </router-link>
        <router-link to="/cart">
          <div class="relative">
            <img class="w-12" src="/cart.png" />
          </div>
          <div
            v-if="cartStore.cart.length > 0"
            class="rounded-full absolute bg-green-500 w-5 h-5 text-white text-sm -right-0 -bottom-
0.5"
          >
```

```
          {{ cartStore.cart.length }}
        </div>
      </router-link>
      <div
        v-if="favoriteStore.FavoriteOpen"
        class="absolute mr-0 top-12 bg-white border-2 rounded-xl z-10 transition"
      >
        <div class="w-80 mt-2 text-left p-4">
          <span class="ml-1 text-slate-600">Улюблене:</span>
          <div>
            <div v-if="favoriteStore.itemsInFavorite.length > 0">
              <div
                class="relative flex mt-4 border-2 rounded-xl hover:border-slate-500 cursor-pointer
p-1"
                v-for="favoriteItem in favoriteStore.itemsInFavorite"
                :key="favoriteItem._id"
                @click="navigateToDetail(favoriteItem._id)"
              >
                <img
                  :src="`https://deadmarket.pp.ua/uploads/${favoriteItem.image}`"
                  class="object-cover h-16 w-20 rounded-xl"
                />

                <div class="ml-5 flex flex-col flex-1">
                  <p class="text-xs">{{ favoriteItem.name }}</p>
                  <p class="text-sm">{{ favoriteItem.price }} грн.</p>
                </div>
                <button @click.stop="favoriteStore.removeFromFavorite(favoriteItem._id)">
                  <img
                    src="/delete.png"
                    class="absolute w-5 h-5 right-2 bottom-2 opacity-20 hover:opacity-100 transition"
                  />
                </button>
              </div>
            </div>
            <div class="ml-1 text-slate-300" v-else>Товари відсутні</div>
          </div>
        </div>
      </div>
    </div>
  </div>
</template>

<script setup>
import { useFavoriteStore } from '../stores/favorites'
import { useRouter } from 'vue-router'
import { useCartStore } from '@/stores/cart'

const favoriteStore = useFavoriteStore()
const cartStore = useCartStore()
const router = useRouter()
```

```
const navigateToDetail = (id) => {
  router.push({ name: 'product-detail', params: { id } })
}
</script>
//ProfilePage.vue
<template>
  <main>
    <div class="sm:flex m-10 z-0">
      <div v-if="userStore.userId">
        <div class="text-3xl mb-5">Мої замовлення:</div>
        <p>{{ userStore.userId }}</p>

        <button
          @click="signOutMethod"
          class="border-2 pl-3 pr-3 bg-black border-black text-white w-full mt-5"
        >
          Вийти з аккаунту
        </button>
      </div>
      <div class="sm:ml-20 sm:w-1/2">
        <ul v-if="sortedOrders.length > 0">
          <li v-for="order in sortedOrders" :key="order._id" class="mb-4 border p-3 relative">
            <div>
              <p><strong>Номер замовлення:</strong> {{ order._id }}</p>
              <p><strong>Усього:</strong> {{ order.total }} грн</p>
              <p><strong>Дата замовлення:</strong> {{ formatDate(order.createdAt) }}</p>
              <p><strong>Статус:</strong> {{ translateStatus(order.status) }}</p>
              <p><strong>Ім’я:</strong> {{ order.customerName }}</p>
              <p><strong>Прізвище:</strong> {{ order.customerSurname }}</p>
              <p><strong>Телефон:</strong> {{ order.customerPhone }}</p>
              <p><strong>Адреса:</strong> {{ order.customerAddress }}</p>
              <div class="mt-3">
                <h4 class="font-bold">Товари:</h4>
                <ul>
                  <li
                    v-for="product in order.products"
                    :key="product.productId"
                    class="flex items-center mt-2 h-24 max-sm:mb-5 max-sm:text-sm"
                  >
                    <img
                      :src="`https://deadmarket.pp.ua/uploads/${product.image}`"
                      alt="Product Image"
                      class="object-cover h-full w-24"
                    />
                    <div class="ml-3 mb-2">
                      <p>
                        <strong>{{ product.name }}</strong>
                      </p>
                      <p>Ціна: {{ product.price }} грн</p>
                      <p>Розмір: {{ product.selectedSize }}</p>
                    </div>
```

```
              </li>
            </ul>
          </div>
          <button
            v-if="order.status !== 'Cancelled'"
            @click="cancelOrder(order._id)"
            class="mt-3 border-2 text-white border-red-500 bg-red-500 p-2 text-sm sm:absolute
right-5 bottom-5 sm:w-40 w-full"
          >
            Відмінити
          </button>
        </div>
      </li>
    </ul>
    <div v-else>
      <p>Немає замовлень для відображення</p>
    </div>
  </div>
 </div>
</main>
</template>

<script setup>
import { getAuth, signOut } from 'firebase/auth'
import { useUserStore } from '@/stores/user'
import router from '@/router'
import { onMounted, computed } from 'vue'
import axios from 'axios'

const userStore = useUserStore()

const signOutMethod = async () => {
 await signOut(getAuth())
 router.push('/login')
}

const cancelOrder = async (orderId) => {
 try {
  await axios.put(`https://deadmarket.pp.ua/orders/${orderId}/cancel`)
  await userStore.fetchUserOrders() // Обновление списка заказов
 } catch (error) {
  console.error('Error cancelling order:', error)
  alert('Error cancelling order')
 }
}

const translateStatus = (status) => {
 const statusTranslations = {
  Pending: 'В очікуванні',
  Shipped: 'Відправлено',
  Delivered: 'Доставлено',
  Cancelled: 'Скасовано'
```

```
  }
  return statusTranslations[status] || status
}

const formatDate = (dateString) => {
  const options = {
    year: 'numeric',
    month: 'long',
    day: 'numeric',
    hour: '2-digit',
    minute: '2-digit'
  }
  return new Date(dateString).toLocaleString('uk-UA', options)
}

const sortedOrders = computed(() => {
  return [...userStore.userOrders].sort((a, b) => new Date(b.createdAt) - new Date(a.createdAt))
})

onMounted(async () => {
  const auth = getAuth()
  const user = auth.currentUser
  if (user) {
    userStore.setUserId(user.uid)
    await userStore.fetchUserOrders()
  } else {
    router.push('/login')
  }
})
</script>

<style scoped></style>
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

на кваліфікаційну роботу бакалавра на тему:

«Розробка вебдодатку для інтернет магазину одягу з автоматизованою системою прийому замовлень »

Студента групи 121-20-2 Фоменка Владислава Олександровича

Керівник економічного розділу
доц. каф. ПЕП та ПУ, к.е.н **Л.В. Касьяненко**

ДОДАТОК В

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файлу	**Опис**
Пояснювальні документи	
ПЗ_Фоменко.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
ПЗ_Фоменко.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
OnlineShop.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_Фоменко.ppt	Презентація кваліфікаційної роботи