

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Вебдодаток для продажу ветеринарних товарів:
функціонал, фільтрація, сортування, допомога»**

Виконав студент групи ПЗ-20-2
КЛІМЧУК Богдан Станіславович

Керівник роботи:
ВЛАСЕНКО Олег Васильович

Рецензент:
КОНЦИДАЙЛО Андрій Михайлович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна ВАКАЛЮК

«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **КЛІМЧУК Богдан Станіславович**

Керівник роботи: **ВЛАСЕНКО Олег Васильович**

Тема роботи: **«Вебдодаток для продажу ветеринарних товарів: функціонал, фільтрація, сортування, допомога»,**

затверджена Наказом закладу вищої освіти від «23» лютого 2024 р., №74/с

Вихідні дані для роботи: **методи та засоби розробки веб-додатків, аналіз проблематики додатку для продажу ветеринарних товарів, використання для розробки веб-додатку вебфреймворку NextJS та NestJS.**

РЕФЕРАТ

Випускна робота бакалавра включає розробку вебдодатку для продажу ветеринарних товарів та пояснювальну записку. Пояснювальна записка містить 83 сторінки, 29 ілюстрацій та 20 таблиць.

Метою роботи є створення веб-додатку з використанням Next.js та Node.js, що покращує доступність ветеринарних товарів для власників домашніх тварин через інтернет. Додаток забезпечує зручність користування та підвищує ефективність процесу продажу товарів, дозволяючи власникам легко знаходити і купувати необхідні товари.

В роботі визначено основні завдання розробки системи, проаналізовано аналоги програмного продукту та обрано засоби реалізації. Проведено розробку структури бази даних, вибрано сучасний стек технологій та розроблено концепцію програмного продукту. Описано програмну реалізацію та результати тестування.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, ОДНОСТОРІНКОВИЙ ДОДАТОК, ТОВАРИ, ТВАРИНИ, ЗРУЧНІСТЬ, ПРОДАЖІ, NEXTJS, NODEJS, OPENAI, НОВА ПОШТА.

					ІПЗ.КР.Б-121-24-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка веб додатку з продажу ветеринарних товарів функціонал, фільтрація, сортування, допомога	Літ.	Арк.	Аркушів
Розроб.		Клімчук Б.С.						
Керівник		Власенко О.В.					3	83
Н. контр.						Житомирська політехніка група ІПЗ-20-2		
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The bachelor's final work includes the development of a web unit for the sale of veterinary goods and an explanatory note. The explanatory note contains 83 pages, 29 illustrations and 20 tables.

The purpose of the work is to create a web application using Next.js and Node.js, which improves the availability of veterinary products for pet owners via the Internet. The application provides convenience of use and increases the efficiency of the process of sale of goods, allowing owners to easily find and buy the necessary goods.

The work identifies the main tasks of system development, analyzes analogues of the software product and selected implementation tools. The database structure was developed, a modern stack of technology has been selected and the concept of software was developed. The software implementation and test results are described.

KEYWORDS: WEB APPLICATION, ONE-PAGE APPLICATION, GOODS, ANIMALS, CONVENIENCE, SALES, NEXT.JS, NODE.JS, OPENAI, MONOBANK, NOVA POST.

					ИПЗ.КР.Б-121-24-ПЗ	Арк.
						4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМІВ ЗАСТОСУВАННЯ ВЕБДОДАТКУ ДЛЯ ПРОДАЖУ ВЕТЕРИНАРНИХ ТОВАРІВ.....	9
1.1. Постановка задачі.....	9
1.2 Аналіз аналогів програмного продукту	11
1.3 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	16
1.3.1. Визначення архітектури клієнтської частини веб-додатку	16
1.3.2. Визначення архітектури серверної частини веб-додатку	18
Висновки до першого розділу.....	20
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	21
2.1 Визначення варіантів використання та структури системи	21
2.2 Розробка бази даних системи.....	30
2.3 Проектування та реалізація алгоритмів роботи додатку	35
2.4 Реалізація функціоналу додатку	39
Висновки до другого розділу:.....	47
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ- ДОДАТКОМ ДЛЯ ПРОДАЖУ ВЕТЕРИНАРНИХ ТОВАРІВ.....	49
3.1 Порядок встановлення та налаштування параметрів додатку	49
3.2 Структура інтерфейсу веб-додатку та порядок роботи	52
3.3 Тестування роботи веб-додатку.....	60
Висновки до третього розділу	64
ВИСНОВКИ.....	66
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	67
ДОДАТКИ.....	69
Додаток А.....	70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

CRUD – основні операції, що використовуються для управління даними (Create , Read , Update , Delete)

DOM – об'єктна модель документа

SPA – односторінковий застосунок

SDK – набір для розробки програмного забезпечення (software development kit)

БД – база даних

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						6

ВСТУП

Актуальність теми полягає у необхідності створення ефективного вебдодатку для продажу ветеринарних товарів з огляду на зростаючий попит на ветеринарні послуги та товари для домашніх улюбленців. В останні роки спостерігається значне зростання кількості власників домашніх тварин, які шукають зручні та надійні способи придбання ветеринарних товарів. Традиційні методи продажу товарів через фізичні магазини поступаються місцем електронній комерції, яка надає можливість швидкого та зручного доступу до широкого асортименту продукції. Розробка спеціалізованого вебдодатку дозволяє не тільки поліпшити обслуговування клієнтів, але й розширити ринок збуту та забезпечити більш ефективне управління запасами і логістикою.

Метою випускної кваліфікаційної роботи бакалавра є розробка вебдодатку для продажу ветеринарних товарів. Для реалізації поставленої мети потрібно виконати наступні завдання:

1. Провести аналіз існуючих рішень для електронної комерції у сфері ветеринарних товарів.
2. Визначити основні вимоги до функціональності вебдодатку.
3. Обрати засоби проектування та розробки програмного забезпечення.
4. Розробити структуру вебдодатку та реалізувати основні модулі.
5. Провести тестування вебдодатку та оцінити його ефективність.

Об'єктом дослідження є процес електронної комерції у сфері ветеринарних товарів, що включає інтерактивні системи управління асортиментом, обслуговування клієнтів, здійснення покупок та логістику доставки.

Предметом дослідження є функціональність вебдодатку для продажу ветеринарних товарів, його архітектура, методи інтеграції з платіжними системами, засоби забезпечення безпеки даних користувачів та оптимізація процесу покупки і доставки товарів. Дослідження також охоплює аналіз переваг та недоліків використання вебдодатків у порівнянні з традиційними методами продажу ветеринарних товарів.

Для розробки веб-додатку було використано вебфреймворки NextJS та NestJS, також для створення швидких і легко керованих API JavaScript було використано Strapi, з об'єктно реляційною системою керування базами даних – PostgreSQL.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						8

РОЗДІЛ 1. АНАЛІЗ НАПРЯМІВ ЗАСТОСУВАННЯ ВЕБДОДАТКУ ДЛЯ ПРОДАЖУ ВЕТЕРИНАРНИХ ТОВАРІВ

1.1. Постановка задачі

Метою кваліфікаційної роботи бакалавра є створення вебдодатку для продажу ветеринарних товарів, що забезпечить функціональність, фільтрацію, сортування товарів та надасть допомогу користувачам у процесі здійснення покупок.

Вебдодаток має на меті поліпшення доступності ветеринарних товарів для власників домашніх тварин через інтернет, забезпечення зручності користування та підвищення ефективності процесу продажу товарів. Це дозволить власникам домашніх тварин легко знаходити та купувати необхідні товари з урахуванням їх потреб.

Основні завдання, що потребують вирішення для досягнення мети, включають:

1. Аналіз ринку та вимог до вебдодатку:

- Провести дослідження існуючих рішень для електронної комерції у сфері ветеринарних товарів.
- Визначити основні вимоги до функціональності, фільтрації, сортування товарів та допомоги користувачам.

2. Проектування архітектури вебдодатку:

- Розробити архітектуру клієнтської та серверної частин вебдодатку.
- Визначити структуру бази даних для зберігання інформації про товари, користувачів та замовлення.

3. Розробка дизайну та користувацького інтерфейсу:

- Створити загальну концепцію дизайну додатку.
- Розробити UI компоненти для зручного користування додатком.
- Забезпечити адаптивність дизайну для різних пристроїв.

4. Реалізація основного функціоналу:

- Розробити модулі для додавання, редагування та видалення товарів.
- Запровадити систему фільтрації та сортування товарів за різними критеріями (ціна, категорія, популярність).
- Створити механізм для надання допомоги користувачам (онлайн-чат).

5. Інтеграція з платіжними системами:

- Забезпечити можливість здійснення безпечних платежів через популярні платіжні системи.
- Реалізувати систему обробки замовлень та відстеження їх статусу.

6. Тестування та оптимізація:

- Провести функціональне тестування всіх модулів вебдодатку.
- Виконати тестування графічного інтерфейсу для забезпечення зручності користування.
- Оптимізувати продуктивність додатку для швидкої роботи навіть при великій кількості користувачів.

Основними етапами розробки вебдодатку є:

1. Розробка дизайну вебсайту:

- Створення загальної концепції дизайну додатку.
- Розробка UI компонентів.
- Розробка дизайну сторінок.

2. Побудова архітектури клієнтської частини додатку:

- Визначення технологій для реалізації клієнтської частини додатку.
- Розробка компонентів інтерфейсу користувача.

3. Побудова архітектури серверної частини додатку:

- Визначення технологій для реалізації серверної частини додатку.
- Розробка API для взаємодії з клієнтською частиною.

4. Розробка системи авторизації та ролей користувачів:

- Реалізація механізму реєстрації та авторизації користувачів.
- Впровадження системи ролей для адміністраторів та користувачів.

5. Розробка основного функціоналу:

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						10

- Створення логіки роботи з товарами (додавання, редагування, видалення).
- Реалізація системи фільтрації та сортування товарів.
- Впровадження механізму для надання допомоги користувачам.

6. Інтеграція з платіжними системами:

- Підключення до популярних платіжних систем.
- Реалізація механізму обробки платежів та замовлень.

7. Тестування додатку:

- Функціональне тестування додатку.
- Тестування графічного інтерфейсу.

Результатом виконання плану дій є створення вебдодатку, який дозволить користувачам зручно та швидко знаходити, фільтрувати, сортувати та купувати ветеринарні товари, отримуючи при цьому необхідну допомогу та підтримку.

1.2 Аналіз аналогів програмного продукту

Розробка вебдодатку з продажу ветеринарних товарів потребує глибокого аналізу конкурентів на ринку.

Одним з таких додатків є Pethouse

Pethouse – популярний український інтернет-магазин ветеринарних товарів. Пропонує широкий вибір кормів, аксесуарів, іграшок та ветеринарних препаратів для домашніх тварин. Відомий зручним інтерфейсом сайту, швидкою доставкою по Україні та регулярними акціями. Магазин також надає професійні консультації щодо догляду за тваринами (рис1.1).

Основними перевагами є:

- Великий асортимент товарів: магазин пропонує широкий вибір товарів для домашніх тварин, включаючи корми, аксесуари, іграшки та ветеринарні препарати.
- Зручний інтерфейс сайту: Інтерфейс сайту є простим і зручним для користувачів, з можливістю фільтрації та сортування товарів.
- Швидка доставка: Магазин забезпечує швидку доставку товарів по всій Україні, що є зручним для клієнтів.

- Постійні акції та знижки: Регулярні акції та знижки роблять покупки вигіднішими.
- Професійні консультації: Магазин пропонує консультації від фахівців щодо вибору товарів та догляду за тваринами.

Недоліками є:

- Ціни: Деякі товари можуть бути дорожчими порівняно з конкурентами.
- Недоліками є Більшість продажів здійснюється через інтернет, що може бути незручним для деяких клієнтів.
- Недоліками є Деякі специфічні продукти можуть бути відсутні в асортименті.

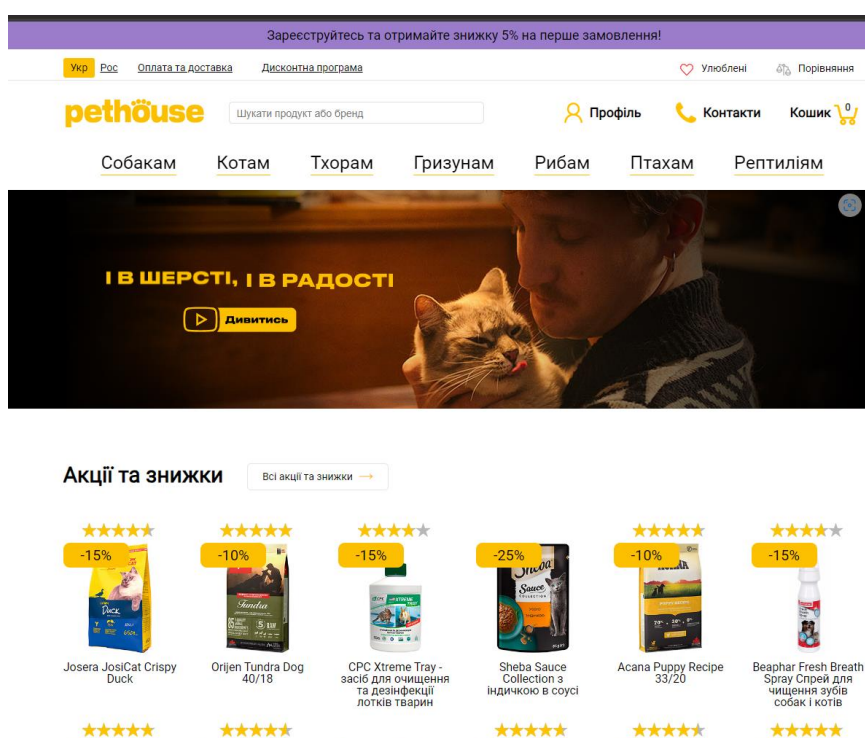


Рис. 1.1. Інтерфейс додатку Pethouse [1]

Е-zoo – український інтернет-магазин, який спеціалізується на товарах для домашніх тварин. Пропонує великий асортимент кормів, аксесуарів, іграшок та ветеринарних препаратів. Магазин відзначається зручним дизайном сайту, швидкою доставкою та програмою лояльності для постійних клієнтів. Регулярно проводить акції та знижки (рис1.2).

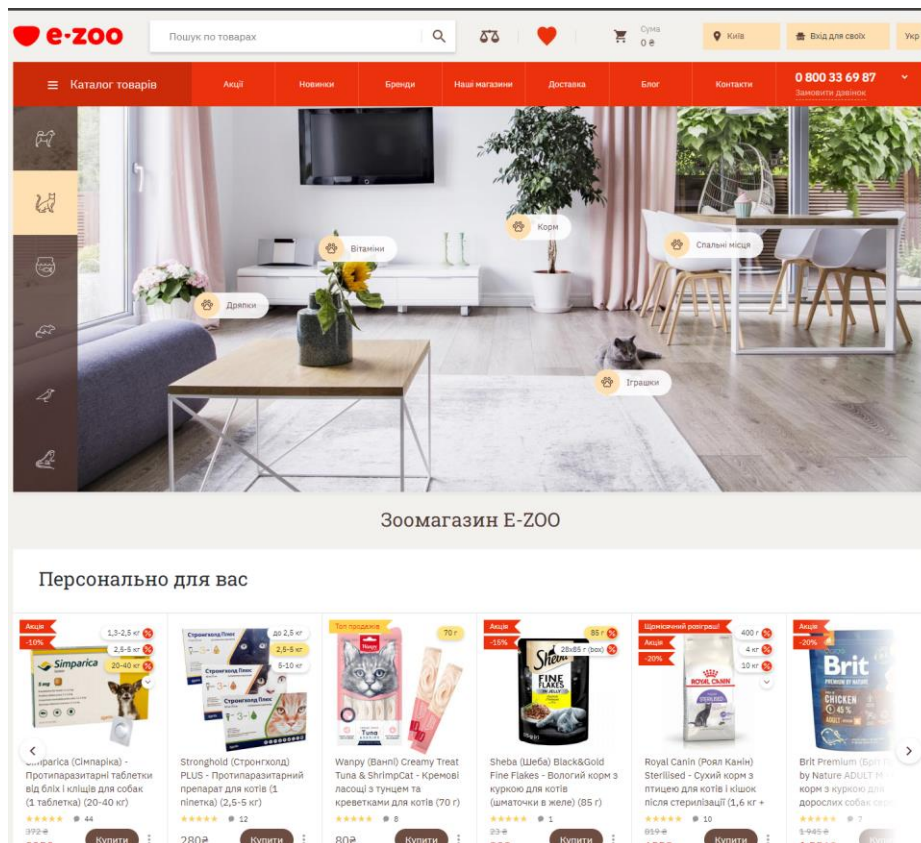


Рис. 1.2. Інтерфейс додатку E-zoo[2]

Основними перевагами є:

- Великий асортимент товарів: Е-zoo пропонує широкий вибір товарів для домашніх тварин, включаючи корми, аксесуари, іграшки, ветеринарні препарати та інші продукти.
- Зручний інтерфейс користувача: Сайт має зручний і простий дизайн, що полегшує навігацію та пошук необхідних товарів.
- Регулярні акції та знижки: Е-zoo часто проводить акції та пропонує знижки, що робить покупки вигіднішими.
- Швидка доставка: Магазин забезпечує швидку доставку товарів по всій Україні, що є зручним для клієнтів.
- Програма лояльності: Існує система бонусів для постійних клієнтів, яка дозволяє економити на майбутніх покупках.

Недоліки:

- Ціни: Деякі товари можуть бути дорожчими порівняно з іншими магазинами.

- Обмежений вибір фізичних магазинів: Всі покупки здійснюються онлайн, що може бути незручним для деяких клієнтів.
- Обмежений асортимент специфічних товарів: Незважаючи на широкий асортимент, деякі специфічні товари можуть бути відсутні.

Zooboом – інтернет-магазин ветеринарних товарів в Україні. Пропонує широкий асортимент кормів, аксесуарів та ветеринарних препаратів для домашніх тварин. Магазин характеризується зручним інтерфейсом, швидкою доставкою по всій Україні та професійними консультаціями щодо вибору товарів. Zooboом також регулярно проводить акції та знижки (рис1.2.3).

Переваги:

- Широкий асортимент товарів: Zooboом пропонує великий вибір товарів для домашніх тварин, включаючи корми, аксесуари, іграшки, ветеринарні препарати та інші продукти.
- Зручний інтерфейс сайту: Інтерфейс сайту є простим і зручним для користувачів, з можливістю фільтрації та сортування товарів.
- Професійні консультації: Zooboом надає консультації від фахівців щодо вибору товарів та догляду за тваринами.
- Швидка доставка: Магазин забезпечує швидку доставку товарів по всій Україні, що є зручним для клієнтів.
- Постійні акції та знижки: Регулярні акції та знижки роблять покупки вигіднішими.

Недоліки:

- Ціни: Деякі товари можуть бути дорожчими порівняно з конкурентами.
- Обмежена кількість фізичних магазинів: Більшість продажів здійснюється через інтернет, що може бути незручним для деяких клієнтів.
- Обмежений асортимент специфічних товарів: Деякі специфічні продукти можуть бути відсутні в асортименті.

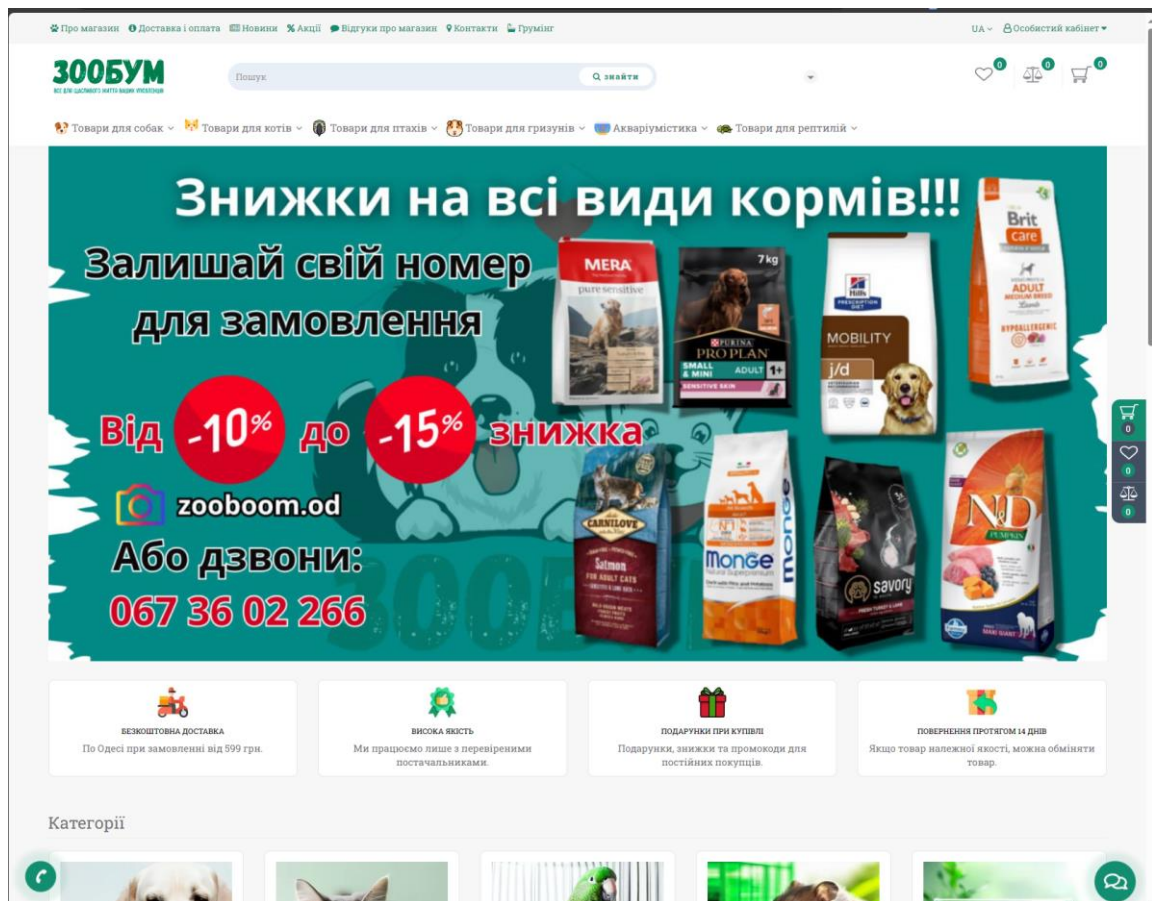


Рис. 1.3. Інтерфейс додатку Зообум[3]

Виходячи з наявних переваг та недоліків, складемо порівняльну таблицю вищезгаданих програмних аналогів (табл. 1.1).

Таблиця 1.1

Характеристика	Pethouse	E-zoo	Zooboom
Перегляд товарів	Зручний	Зручний	Зручний
Пошук та фільтрація	Є пошук фільтрація, але немає фільтрації за ціною	Є пошук фільтрація	Є пошук фільтрація
Адаптивність інтерфейсу	Адаптивний	Адаптивний	Адаптивний
Зручність використання	Зручний	Зручний	Зручний

Наявність специфічних товарів	Деякі специфічні відсутні	Деякі специфічні відсутні	Деякі специфічні відсутні
Дизайн	Непоганий, але загромаджений	Чудовий	Чудовий

Провівши аналіз готових рішень стало зрозуміло, що існуючі інтернет-магазини ветеринарних товарів в Україні, такі як Pethouse, E-zoo, та Zooboom, мають низку переваг, включаючи широкий асортимент товарів, зручний інтерфейс, програми лояльності, швидку доставку та професійні консультації. Однак, їхні недоліки, такі як високі ціни, обмежена кількість фізичних магазинів і відсутність деяких специфічних товарів, створюють можливості для поліпшення.

Було вирішено створити власний програмний продукт для продажу ветеринарних товарів, щоб запропонувати унікальні функції та інноваційні рішення, забезпечити оптимізацію і швидкість роботи, гнучкість та масштабованість, а також побудувати власний бренд і контроль якості. Це дозволить краще задовольнити потреби клієнтів і забезпечити конкурентоспроможність на ринку.

1.3 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Веб-додаток буде реалізований за клієнт-серверною архітектурою[22]. Серверна частина, що функціонуватиме у вигляді API, буде розроблена з використанням сучасного фреймворку NestJS[4]. Для клієнтської частини буде застосовано фреймворк NextJS, який базується на бібліотеці ReactJS. В якості сховища даних використовуватиметься база даних PostgreSQL.

1.3.1. Визначення архітектури клієнтської частини веб-додатку

Для розробки клієнтської частини вебдодатку було обрано фреймворк NextJS[5]. Це сучасний інструмент, побудований на основі ReactJS, який

дозволяє створювати високопродуктивні та SEO-оптимізовані вебдодатки завдяки підтримці серверного рендерингу (SSR) та статичної генерації (SSG).

Основні переваги NextJS:

Server-Side Rendering (SSR): Дозволяє відтворювати сторінки на сервері, що збільшує швидкість завантаження сторінок та покращує їх SEO-оптимізацію[6].

Static Site Generation (SSG): Генерація статичного контенту під час збірки додатку, що зменшує навантаження на сервер та збільшує швидкість завантаження сторінок[7].

Client-Side Rendering (CSR): Підтримка клієнтського відтворення для створення динамічного контенту та інтерактивних елементів[8].

TypeScript: Вбудована підтримка TypeScript, що забезпечує більш надійну та зручну розробку завдяки строго типізованому коду[9].

Можливості розширення: Підтримка багатьох різних плагінів та модулів для розширення функціональності та забезпечення більшої гнучкості розробки.

Для додаткової функціональності клієнтської частини були використані наступні бібліотеки[10-19]:

TypeScript: Забезпечує надійність та зручність розробки завдяки строго типізованому коду.

ShadCN: Бібліотека що надає велику кількість готових headless компонентів що суттєво пришвидшує розробку і дозволяє дуже гнучко їх стилізувати.

Tanstack-query: Бібліотека для управління станом та обробки запитів до серверу, що дозволяє оптимізувати роботу з даними. Також автоматично кешує дані що зменшує кількість завантажень що в свою чергу покращує користувацький досвід.

Tailwind: CSS-фреймворк для швидкого створення стильних та адаптивних інтерфейсів. Він надає набір утилітарних класів, що дозволяє швидко створювати стильні та адаптивні інтерфейси без написання власного CSS.

OpenAI: Інтеграція з API OpenAI для реалізації допоміжних функцій та автоматизації деяких задач. Використовується для побудови чату підтримки[20].

Для отримання інформації про населені пункти України було використано *API нової пошти*[21]

1.3.2. Визначення архітектури серверної частини веб-додатку

Для розробки серверної частини вебдодатку було обрано фреймворк **NestJS**, який є потужним інструментом для створення серверних застосунків на Node.js з використанням TypeScript.

Основні переваги NestJS:

TypeScript: Забезпечує надійність та зручність розробки завдяки строго типізованому коду.

Модульність: Підтримка модульної архітектури, що дозволяє легко організовувати код та зменшує його складність.

Dependency Injection: Вбудований механізм впровадження залежностей, що забезпечує високу розширюваність додатків.

WebSocket: Вбудована підтримка WebSocket для розробки реального часу додатків.

Тестованість: Модульна структура та вбудовані механізми для тестування роблять тестування додатків легким та зручним.

Інтеграція: Може працювати з багатьма іншими інструментами, такими як Swagger для написання документації, що спрощує процес розробки.

Для управління контентом було використано **CMS Strapi**, яка дозволяє швидко створювати API та управляти даними через зручний інтерфейс.

Переваги Strapi:

Гнучкість: Легко налаштовується для будь-яких потреб завдяки модульній архітектурі.

Зручність: Зручний адміністративний інтерфейс для управління контентом.

Безпека: Вбудовані засоби для забезпечення безпеки даних.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						18

Для зберігання даних було обрано реляційну базу даних **PostgreSQL**, яка забезпечує високу надійність та продуктивність при роботі з великими обсягами даних.

Переваги PostgreSQL[23]:

Масштабованість: Підтримує роботу з великими обсягами даних та високонавантаженими системами.

Надійність: Забезпечує високий рівень захисту даних та підтримує складні запити.

Інтеграція: Легко інтегрується з іншими інструментами та фреймворками.

Вимоги до апаратного забезпечення

Для забезпечення стабільної та швидкої роботи вебдодатку, необхідно врахувати наступні вимоги до апаратного забезпечення:

1. Сервер:

- Процесор: Багатоядерний процесор (рекомендовано 4 ядра і більше).
- Оперативна пам'ять: Мінімум 8 ГБ оперативної пам'яті, рекомендовано 16 ГБ.
- Накопичувач: SSD-диск з мінімум 256 ГБ вільного простору.
- Мережа: Високошвидкісне підключення до інтернету з мінімальною затримкою.

2. Клієнтські пристрої:

- Процесор: Сучасний багатоядерний процесор.
- Оперативна пам'ять: Мінімум 4 ГБ оперативної пам'яті.
- Браузер: Останні версії браузерів, таких як Google Chrome, Mozilla Firefox, Microsoft Edge.
- Мережа: Стабільне підключення до інтернету.

Використання вище описаних інструментальних засобів та відповідного апаратного забезпечення забезпечить високу продуктивність, надійність та зручність використання вебдодатку для продажу ветеринарних товарів.

Висновки до першого розділу.

Було проведено аналіз предметної області, в результаті якого визначено основні вимоги до платформи для продажу ветеринарних товарів, основною задачею якої є забезпечення користувачів можливістю зручно знаходити та купувати необхідні товари.

В ході дослідження аналогічних та схожих за концепцією продуктів були проаналізовані та порівняні різні платформи з різними принципами роботи, основним функціоналом яких є пошук, фільтрація та сортування товарів, а також надання допомоги користувачам.

На основі цього були сформовані ключові вимоги для власної платформи. Відповідно до цих вимог та критеріїв було сформовано набір основних технологій та засобів реалізації для розробки вебдодатку.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Визначення варіантів використання та структури системи

Для забезпечення безперервного виконання необхідних функцій і відповідності встановленим стандартам і специфікаціям наведено високорівневі вимоги, які повинна задовольняти веб-орієнтована система планування для продажу ветеринарних товарів. Ці вимоги допомагають забезпечити надійну роботу системи та відповідність усім необхідним стандартам і специфікаціям.

Бізнес вимоги:

Збільшення обсягів продажу ветеринарних товарів: Вебдодаток повинен сприяти зростанню продажів за рахунок зручного інтерфейсу та широкого асортименту товарів. Система повинна забезпечувати збільшення кількості постійних клієнтів через програми лояльності знижки та індивідуальні пропозиції.

Оптимізація бізнес-процесів: Вебдодаток повинен автоматизувати процеси обробки замовлень, управління складом та взаємодії з постачальниками, що дозволить знизити витрати та підвищити ефективність роботи. Система повинна надавати можливість моніторингу та аналізу ключових показників діяльності для прийняття обґрунтованих бізнес-рішень.

Вимоги користувачів

Внутрішній користувач – адміністратор

1. Управління каталогом ветеринарних товарів.
2. Управління категоріями та підкатегоріями товарів.
3. Управління адресами магазинів.
4. Спілкування з клієнтами та постачальниками.
5. Підтвердження та модерація нових замовлень

Внутрішній користувач – клієнт

1. Реєстрація та авторизація.

2. Відновлення втраченого паролю.
3. Зв'язок зі службою підтримки
4. Спілкування з консультантом для отримання допомоги.
5. Перегляд каталогу товарів.
6. Пошук товарів за категоріями та фільтрами.
7. Додавання товарів до кошика.
8. Оформлення та відстеження замовлень.
9. Перегляд історії покупок.
10. Редагування особистої інформації

Внутрішній користувач – гість

1. Перегляд каталогу товарів.
2. Пошук товарів за категоріями та фільтрами.
3. Перегляд детальної інформації про товари.
4. Реєстрація для оформлення замовлень.
5. Зв'язок зі службою підтримки
6. Спілкування з консультантом для отримання допомоги.

Функціональні вимоги

1. *Авторизація користувачів:*
 - Реєстрація через електронну пошту.
 - Авторизація за допомогою логіна та пароля.
2. *Управління каталогом товарів:*
 - Додавання, редагування та видалення товарів.
 - Додавання та редагування адрес магазинів.
 - Фільтрація та сортування товарів за різними критеріями.
3. *Кошик та оформлення замовлень:*
 - Додавання товарів до кошика.
 - Редагування товарів в кошику.
 - Оформлення замовлення з можливістю вибору способу доставки та оплати.
 - Підтвердження та відстеження статусу замовлення.

Нефункціональні вимоги

1. Сприйняття:

- Час, потрібний для освоєння вебсайту – від 3 до 5 хвилин, для досвідчених користувачів менше 2 хвилин.
- Час відповіді вебсайту на дії користувача не повинен перевищувати 2 секунд.
- Інтерфейс вебсайту повинен бути інтуїтивно зрозумілим та зручним для користувача, та не вимагати додаткової підготовки.

2. Надійність:

- Вебсайт має бути доступний цілодобово.
- Система повинна забезпечувати збереження даних навіть у випадку збою.

3. Продуктивність:

- Сайт повинен витримувати одночасне використання великою кількістю користувачів.
- Система повинна забезпечувати швидке завантаження сторінок та обробку запитів.

4. Безпека:

- Захист персональних даних користувачів.
- Використання SSL для шифрування даних.
- Регулярне оновлення системи для захисту від вразливостей.

Після аналізу вимог користувачів і для наглядної демонстрації взаємодії між елементами додатку було визначено варіанти використання додатку для продажу ветеринарних товарів. Для зручного представлення та опису визначених варіантів використання було створено діаграму використання програмної системи (рис 2.1). Це дало змогу чітко визначити та описати всі конкретні сценарії використання програмної системи, що суттєво зменшить кількість помилок при подальшому проектуванні та розробці програмної системи.

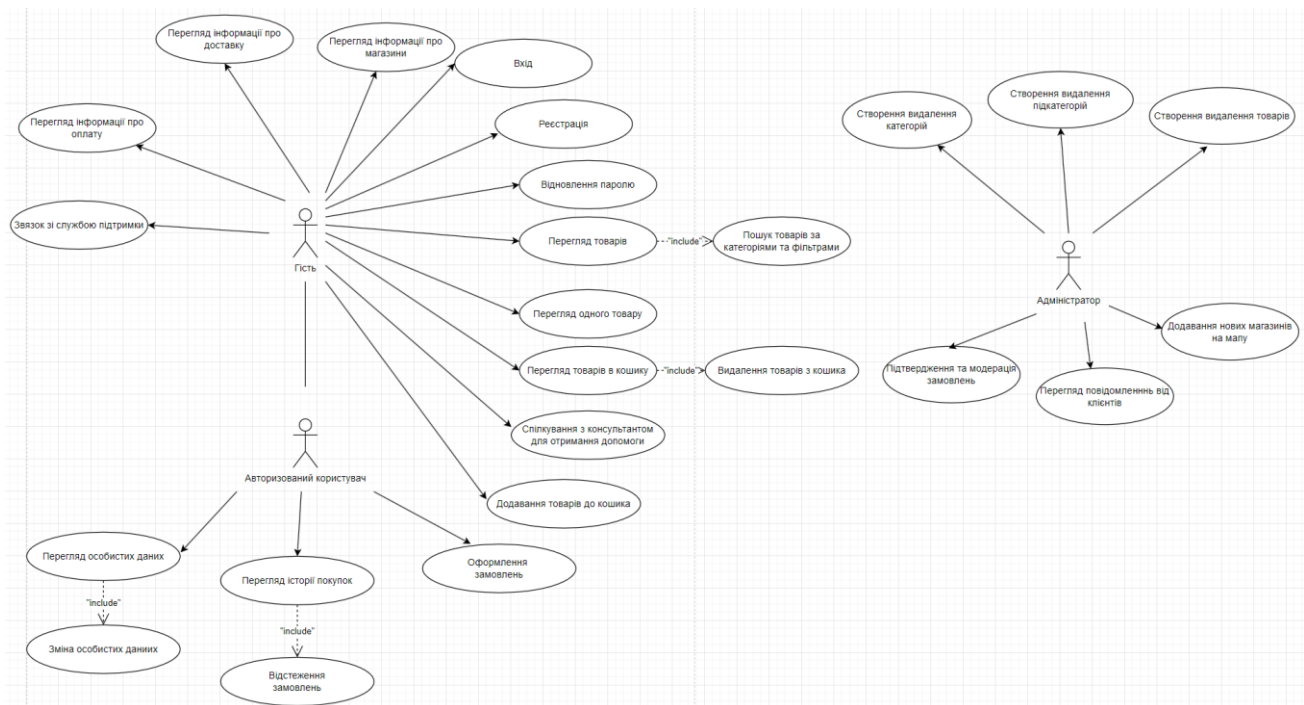


Рис. 2.1. Діаграма варіантів використання

Опис основних варіантів використання наведено нижче. (табл 2.1. – 2.10).

Опис прецеденту «Вхід»

Таблиця 2.1.

Назва варіанту використання	Вхід
Основні актори	Гість
Короткий опис	Гість може увійти в раніше створений обліковий запис для доступу до розширених функцій додатку.

Опис прецеденту «Перегляд товарів»

Таблиця 2.2.

Назва варіанту використання	Перегляд товарів
Основні актори	Гість, Авторизований користувач
Короткий опис	Гість та авторизований користувач можуть переглядати доступні ветеринарні товари, сортувати їх та фільтрувати за різними параметрами.

Опис прецеденту «Перегляд одного товару»

Таблиця 2.3.

Назва варіанту використання	Перегляд одного товару
Основні актори	Гість, Авторизований користувач
Короткий опис	Гість та авторизований користувач можуть вибрати окремий товар з каталогу для перегляду детальної інформації. Це включає детальний опис товару, його характеристики а також фото товару. Користувачі також можуть побачити ціну товару, та мають можливість додати товар до кошика.

Опис прецеденту «Додавання товарів до кошика»

Таблиця 2.4.

Назва варіанту використання	Додавання товарів до кошика
Основні актори	Гість, Авторизований користувач
Короткий опис	Гість та авторизований користувач можуть додавати товари до кошика обравши необхідну кількість товару та натиснувши на необхідну кнопку зі сторінки «Каталог» або перейшовши на сторінку з детальною інформацією про товар

Опис прецеденту «Перегляд товарів в кошику»

Таблиця 2.5.

Назва варіанту використання	Перегляд товарів в кошику
Основні актори	Гість, Авторизований користувач
Короткий опис	Гість та авторизований користувач можуть переглядати товари, додані в кошик, редагувати їх кількість або видаляти непотрібні

	позиції або видалити всі раніше додані до кошика позиції.
--	---

Опис прецеденту «Спілкування з консультантом для отримання допомоги»

Таблиця 2.6.

Назва варіанту використання	Спілкування з консультантом для отримання допомоги
Основні актори	Гість, Авторизований користувач
Короткий опис	Гість та авторизований користувач можуть звернутися до служби підтримки використовуючи форму зворотного зв'язку, або використавши чат побудований на основі моделі штучного інтелекту для отримання порад або допомоги щодо вибору товарів.

Опис прецеденту «Перегляд особистих даних»

Таблиця 2.7.

Назва варіанту використання	Перегляд особистих даних
Основні актори	Авторизований користувач
Короткий опис	Авторизований користувач може переглядати та редагувати свої особисті дані, такі як ім'я, адреса та контактна інформація.

Опис прецеденту «Перегляд історії покупок»

Таблиця 2.8.

Назва варіанту використання	Перегляд історії покупок
Основні актори	Авторизований користувач
Короткий опис	Авторизований користувач може переглядати історію своїх покупок, дізнаватися про статус замовлень та повторно замовляти товари.

Опис прецеденту «Перегляд інформації про доставку»

Таблиця 2.9.

Назва варіанту використання	Перегляд інформації про доставку
Основні актори	Гість, Авторизований користувач
Короткий опис	Гість та авторизований користувач можуть ознайомитися з умовами доставки, доступними службами доставки та приблизними термінами виконання замовлень.

Опис прецеденту «Перегляд інформації про оплату»

Таблиця 2.10.

Назва варіанту використання	Перегляд інформації про оплату
Основні актори	Гість, Авторизований користувач
Короткий опис	Гість та авторизований користувач можуть ознайомитися з доступними способами оплати, умовами повернення коштів та іншими фінансовими аспектами.

Аналіз функціональних вимог дозволив визначити такі сутності, які забезпечать програмну реалізацію додатку а саме Order, User, Basket, Authorization, Product, SubCategory, Category, ShopAddresses, Payment. Для зручного представлення та опису сутностей було створено відповідну діаграму класів (рис. 2.2). Після чого було проаналізовано зв'язки між сутностями, їх поля та методи що дало змогу описати модулі які необхідно створити для реалізації функціоналу пов'язаного з продуктами, фільтрацією та сортуванням товарів, категоріями, користувачами та корзиною користувача. Після описання даних модулів було спроектовано базу даних. Враховуючи потреби та поставлені задачі було вирішено що для реалізації необхідного функціоналу найкращим варіантом буде використання реляційної бази даних.

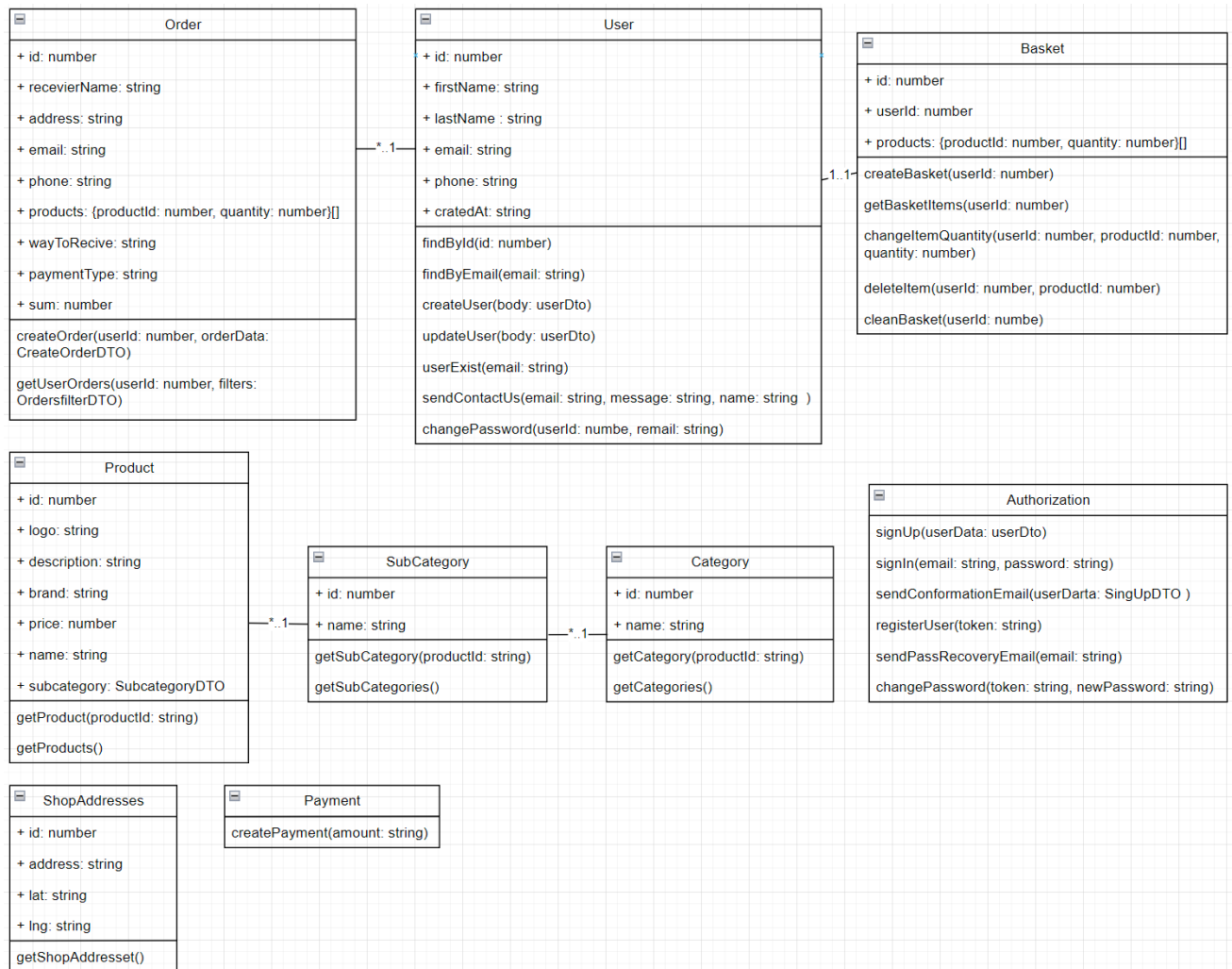


Рис. 2.2. Діаграма класів

User – клас який відповідає за користувача, має наступні методи:

- **createUser()** – метод що відповідає за створення і збереження користувача в базі даних.
- **updateUser()** – метод що відповідає за оновлення даних існуючого користувача.
- **sendContactUs()** – метод що відповідає за оновлення даних існуючого користувача.
- **userExist()** – метод що перевіряє чи є такий користувач в базі даних.
- **changePassword()** – метод що відповідає за зміну пароля в існуючого користувача.
- **findById()** – метод що відповідає за пошук користувача за його ID.

- findByEmail() – метод що відповідає за пошук користувача за його електронною поштою.

Basket – клас який відповідає за роботу з корзиною користувача, має наступні методи:

- createBasket() – метод що відповідає за створення нової корзини та збереження її в базі даних на етапі реєстрації нового користувача.
- getBasketItems() – метод що відповідає за отримання товарів які були додані до корзини.
- changeItemQuantity() – метод що відповідає за додавання нових товарів до корзини та зміну кількості товарів що раніше були додані користувачем до кошика.
- deleteItem() – метод що відповідає за видалення раніше доданого товару з корзини.
- cleanBasket () – метод що відповідає за видалення всіх товарів з корзини що були додані раніше.

Product – клас який відповідає за роботу з товарами, має наступні методи:

- getProduct() – метод що відповідає за отримання детальної інформації про обраний товар.
- getProducts() – метод що відповідає за отримання товарів, відповідно до фільтрів які обрав користувач.
- Аналогічно адресам, створення видалення редагування товарів відбувається в панелі адміністратора створеній з використанням StrapiCMS, тому потреби в окремих методах немає.

SubCategory – клас який відповідає за роботу з підкатегоріями:

- getSubCategory() – метод що відповідає за отримання детальної інформації про одну підкатегорію.
- getSubCategories() – метод що відповідає за отримання всіх підкатегорій.
- Аналогічно товарам, створення видалення редагування підкатегорій відбувається в адмін панелі, тому потреби в окремих методах немає.

2.2 Розробка бази даних системи

Для збереження файлів та даних у додатку використовується PostgreSQL. Це потужна система управління базами даних, заснована на реляційній моделі. Вона забезпечує високий рівень надійності, масштабованості та відповідності стандартам SQL. Використання PostgreSQL у проекті обумовлене її можливістю обробляти великі обсяги даних, підтримкою складних запитів і транзакцій, а також розширюваністю, що дозволяє додавати нові функції без порушення роботи додатку.

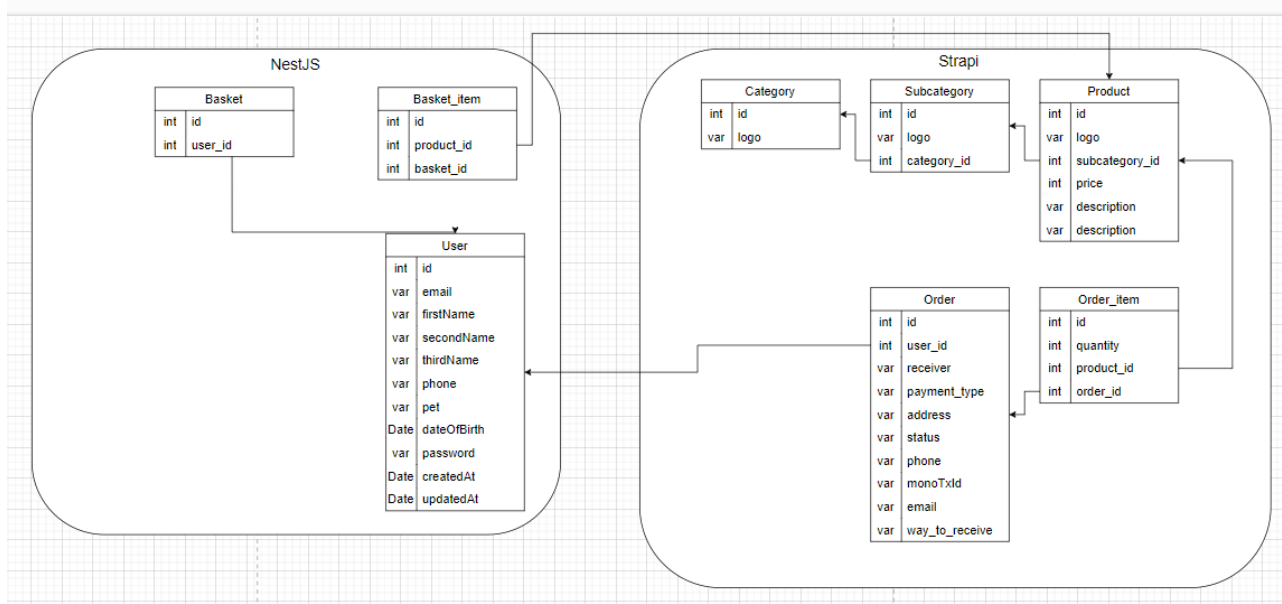


Рис. 2.3. Діаграма взаємодії таблиць в базі даних

База даних системи складається з п'яти таблиць:

1. Users – таблиця, яка зберігає данні користувачів;
2. Basket - таблиця, яка зберігає корзину користувача;
3. Basket item - таблиця, яка відповідає за зберігання доданих про додані до корзини товари
4. Orders - таблиця, яка відповідає за зберігання замовлень користувачів
5. Order item - таблиця, яка відповідає за збереження товарів які були додані до замовлення.
6. ShopAddresses - таблиця, яка відповідає за збереження адрес магазинів.
7. Products - таблиця, яка відповідає за збереження всіх доданих товарів.

8. Subcategory - таблиця, яка відповідає за збереження доданих підкатегорій

9. Category - таблиця, яка відповідає за збереження доданих категорій

Опис кожної з таблиць бази даних наведений у таблицях 2.11 – 2.19.

Таблиця 2.11

Таблиця User

Поле	Тип	Призначення
Id	integer	Ідентифікатор
Email	varchar	Email користувача
firstName	varchar	Ім'я користувача
secondName	varchar	Прізвище користувача
thirdName	varchar	По батькові користувача
phone	varchar	Номер телефону користувача
pet	varchar	Назва домашніх улюбленців користувача
dateOfBirth	date	Дата народження
password	varchar	Захешований пароль користувача
createdAt	timestamp	Дата реєстрації
updatedAt	timestamp	Дата останнього оновлення

Таблиця 2.12

Таблиця Basket

Поле	Тип	Призначення
id	integer	Ідентифікатор
user_id	integer	Ідентифікатор користувача якому належить корзина

Таблиця Basket item

Поле	Тип	Призначення
id	integer	Ідентифікатор
itemId	int	Ідентифікатор доданого товару
quantity	int	Кількість доданого товару
basket_id	integer	Ідентифікатор корзини до якої належить доданий продукт

Таблиця Orders

Поле	Тип	Призначення
id	integer	Ідентифікатор
userId	integer	Ідентифікатор користувача якому належить замовлення
recevier	varchar	Ім'я отримувача
address	varchar	Адреса доставки
status	varchar	Статус замовлення
phone	varchar	Номер телефону отримувача
payment_type	varchar	Спосіб оплати замовлення
monoTxId	varchar	Ідентифікатор транзакції у випадку якщо оплата була проведена онлайн
email	varchar	Електронна пошта

		отримувача
way_to_receive	varchar	Спосіб отримання замовлення
order_items	integer	Ідентифікатор таблиці в якій зберігаються додані до замовлення товари

Таблиця 2.15

Таблиця Order item

Поле	Тип	Призначення
id	integer	Ідентифікатор
quantity	integer	Кількість певного товару
product_id	integer	Ідентифікатор товару
order_id	integer	Ідентифікатор замовлення до якого належать ці товари

Таблиця 2.16

Таблиця ShopAddresses

Поле	Тип	Призначення
id	integer	Ідентифікатор
address	varchar	Адреса магазину
latitude	decimal	Широта
longitude	decimal	Довгота

Таблиця 2.17

Таблиця Products

Поле	Тип	Призначення
id	integer	Ідентифікатор
name	varchar	Назва товару
logo	varchar	Посилання на фото

price	decimal	Ціна товару
description	varchar	Детальний опис товару
brand	varchar	Бренд
subcategory	integer	Ідентифікатор підкатегорії до якої належить товар
category	integer	Ідентифікатор категорії до якої належить товар

Таблиця 2.18

Таблиця SubCategories

Поле	Тип	Призначення
id	integer	Ідентифікатор
category	integer	Назва підкатегорії
name	varchar	Ідентифікатор категорії до якої належить підкатегорія

Таблиця 2.19

Таблиця Categories

Поле	Тип	Призначення
id	integer	Ідентифікатор
logo	varchar	Посилання на фото категорії
name	varchar	Назва категорії

Ці таблиці взаємодіють між собою, забезпечуючи зберігання, управління та обробку всіх необхідних даних для ефективного роботи додатку. Оскільки в SQL немає можливості створювати масиви було створено декілька проміжних таблиць, Basket item та Order item. Також варто зазначити що таблиця ShopAddresses немає зв'язків оскільки використовується лише для збереження

адрес магазинів які ні до чого іншого не прив'язані, тому і не була зображена на діаграмі взаємодії.

2.3 Проектування та реалізація алгоритмів роботи додатку

Для вебдодатку з продажу ветеринарних товарів, написаного на NextJS, використовується алгоритм клієнт-серверного відтворення сторінок з використанням односторінкової архітектури (SPA) для клієнтської частини. Загальний алгоритм роботи додатку зображено на рисунку 2.3.1

При переході на сторінку додатку або відкритті додатку, сервер NextJS генерує та надсилає користувачеві актуальну версію сторінки з попередньо згенерованим HTML-кодом. Це забезпечує швидке початкове завантаження сторінки та покращує SEO-оптимізацію.

На етапі серверного промальовування NextJS може виконувати необхідні запити до API для отримання початкових даних, в нашому випадку відбувається перевірка авторизації користувача. Після завантаження сторінки у браузері користувача, React бере на себе управління додатком, роблячи його динамічним та інтерактивним.

Клієнтська частина проводить ініціалізацію стану додатку на основі початкових даних, отриманих з серверу. Залежно від дій користувача (перегляд товарів, додавання до кошика, оформлення замовлення тощо), клієнтська частина надсилає відповідні запити до серверу через API. Наприклад при додаванні товару до кошика, надсилається запит на створення або оновлення кошика у базі даних.

Сервер обробляє отримані запити, взаємодіє з базою даних та іншими сервісами для отримання необхідної інформації або збереження змін. Після обробки запиту сервер повертає відповідь з необхідними даними або підтвердженням успішного виконання операції. Після отримання відповіді клієнтська частина оновлює UI відповідно до відповіді яку надіслав сервер. Наприклад, після успішного додавання товару до кошика, клієнтська частина оновлює відображення кошика та кількість товарів у ньому

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						35

Користувач може переміщуватися між сторінками додатку без повного перезавантаження сторінки завдяки можливостям NextJS та React. Додаток динамічно завантажує та відображає необхідні компоненти та дані, забезпечуючи швидку та плавну навігацію

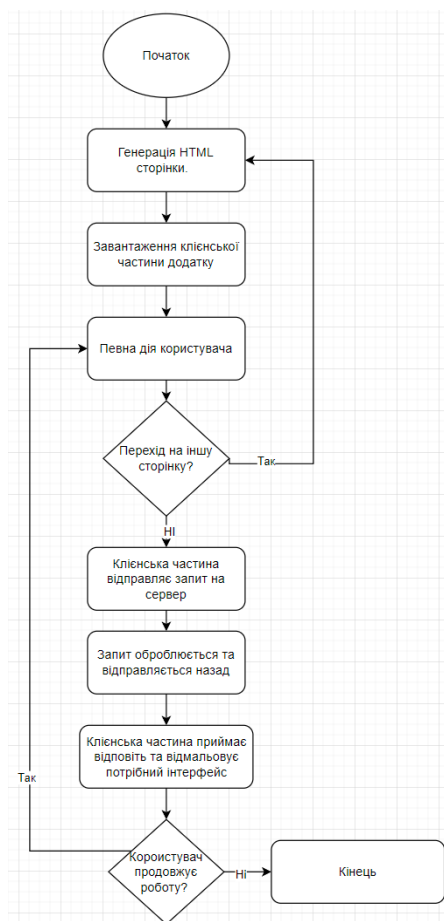


Рисунок 2.4 – Загальний алгоритм роботи додатку

Склавши загальний алгоритм додатку, необхідно розробити алгоритми для окремих випадків. Для представлення таких сценаріїв зазвичай використовуються діаграми діяльності (activity diagrams). Ці діаграми можуть додатково пояснити деякі дії у додатку що в подальшому спростить розробку додатку та зменшить кількість посилок при проектуванні архітектури додатку. Для прикладу було створено декілька діаграм а саме «Авторизація користувача» (рисунок 2.4), та «Робота з корзиною» (рисунок 2.5) , оскільки ці дії пов’язані між собою.

При завантаженні перевіряється чи користувач авторизований і в залежності від цього генерується відповідний UI. Якщо користувач не авторизований йому відкривається доступ до сторінки Login. Після успішної авторизації API створює та повертає токен, далі сервер NextJS зберігає його до «HTTP only» cookies. Логіка роботи з токенами відбувається на стороні серверу, що в свою чергу забезпечує надійний захист токенів та даних користувачів(рисунок 2.4).

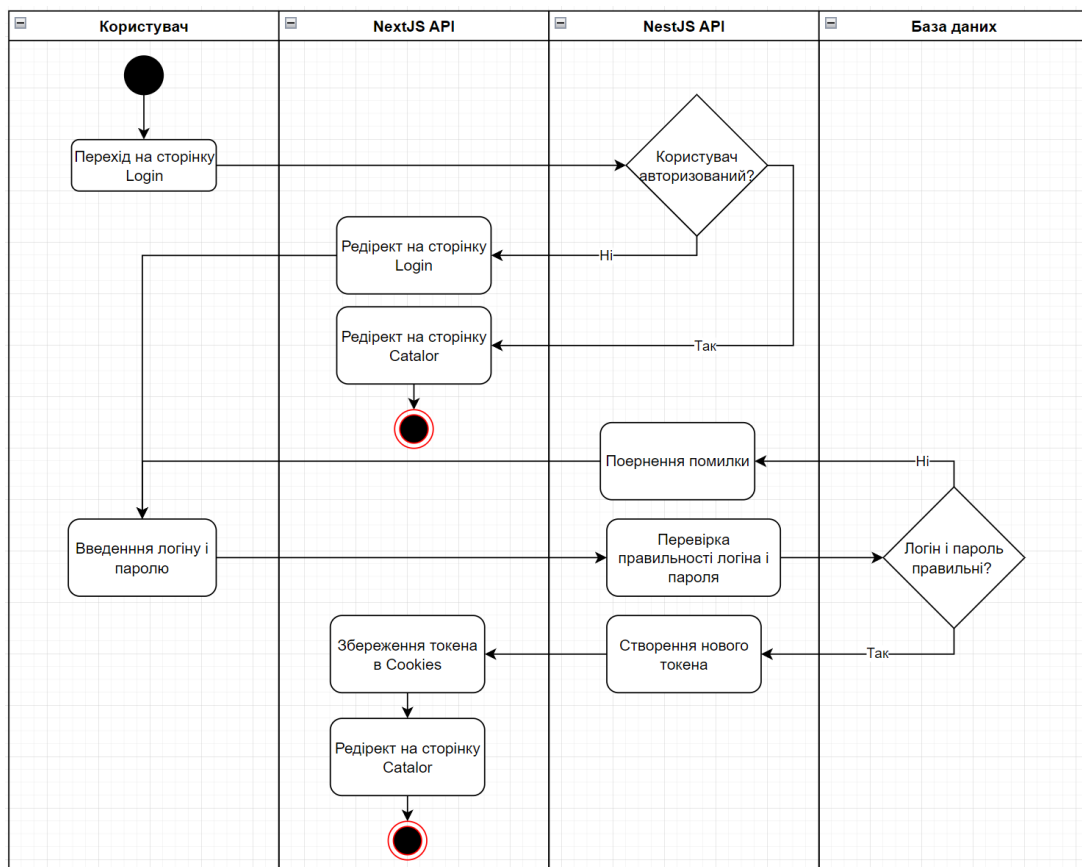


Рисунок 2.5 – Діаграма діяльності «Авторизація користувача»

Після успішної авторизації користувача направляє на сторінку «catalog» . На цій сторінці користувач може переглядати товари або перейти на сторінку з детальною інформацією про товар натиснувши та картку товару.

Товар можна додати до кошика зі сторінки catalog або зі сторінки товару. Якщо обраного товару в кошику немає товар та його кількість додається до списку, якщо такий товар вже було додано раніше, змінюється його кількість. В

залежності від того авторизований користувач чи ні дані зберігаються по різному.

У випадку якщо користувач не авторизований дані зберігаються лише на стороні клієнта. Після успішної авторизації користувача відбувається синхронізація даних збережених в базі даних і тих що зберігаються на стороні клієнта. Після цього всі внесені зміни будуть автоматично зберігатися в базі даних.

Якщо на стороні клієнта є збережені продукти а в базі даних в корзині користувача немає жодного товару, дані з клієнта зберігаються в базу даних. У випадку якщо в базі даних в корзині користувача є продукти ці дані записуються на стороні клієнта (рисунок 2.5).

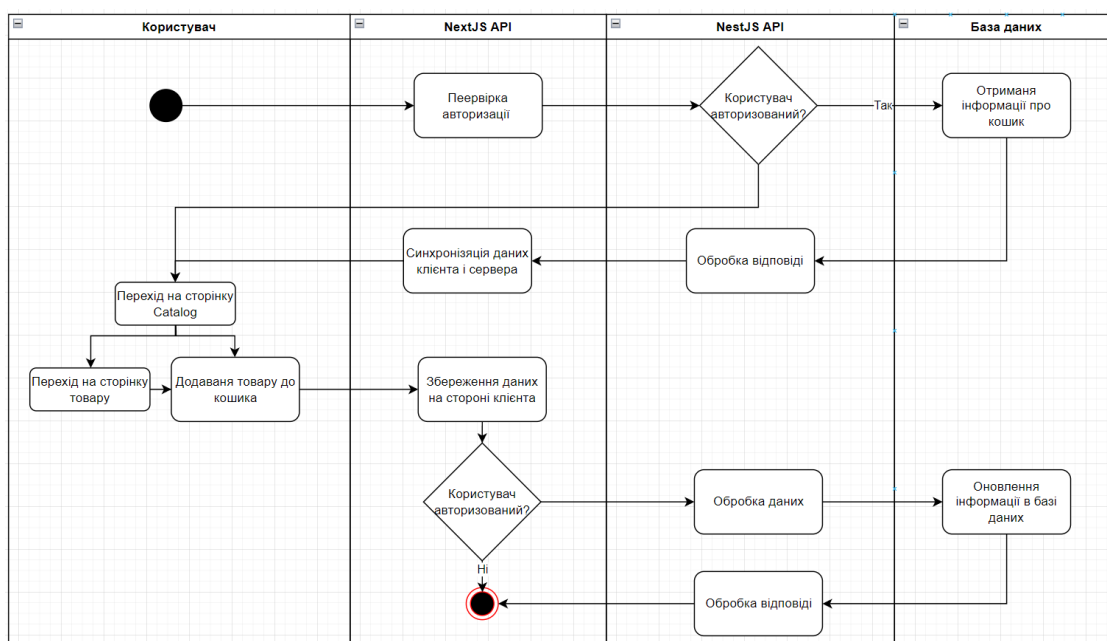


Рисунок 2.6 – Діаграма діяльності «Робота з корзиною»

Після того як користувач обрав потрібні товари він може перейти на сторінку оформлення замовлення.

Після того як користувач введе правильну інформацію та вибере спосіб оплати він отримає можливість оформити замовлення. Наявні два способи оплати, а саме «Оплата при отриманні» та «Оплатити зараз».

Відстежувати статус замовлення на відповідній сторінці. Також є можливість фільтрації замовлень за обраний період часу або за конкретною датою.

2.4 Реалізація функціоналу додатку

Додаток буде складатися з клієнтської частини, серверної частини та панелі адміністратора, для кожної частини буде створено окремий додаток.

Серверна частина додатку була розроблена з використанням фреймворка NestJS. Вхідною точкою в додаток є файл main.ts, а саме функція bootstrap();

В цій функції створюється об'єкт додатку і налаштовуються його параметри, повний код файлу знаходиться в додатку А:

```
const app = await NestFactory.create(AppModule, {
  logger: ['log', 'error', 'warn', 'debug', 'verbose'],
  rawBody: true,
});

app.useGlobalPipes(new ValidationPipe({ transform: true }));
app.enableCors();
const logger = new Logger('Bootstrap');
...
await app.listen(PORT, () => {
  logger.log(`Server started on ${PORT}`);
});
```

В каталозі з цим файлом присутній інший файл, а саме config.ts. Він відповідає за налаштування серверу та його завантаження.

```
export const mailerConfig: MailerOptions = {
  transport: {
    host: 'sandbox.smtp.mailtrap.io',
    port: 2525,
    auth: {
      user: process.env.MAILER_USER || 'c2222222222222',
      pass: process.env.MAILER_PASSWORD || '6b666666666666',
    },
  },
};

export default () => ({
  env: process.env.ENV || 'development',
  port: Number(process.env.PORT) || 3001,
  app: {
    url: process.env.APP_URL || 'https://app.codlo.pp.ua',
  },
});
```

```
});
```

В файлі `app.module.ts` підключаються всі модулі що необхідні для роботи веб додатку.

```
@Module({
  imports: [
    TypeOrmModule.forRoot({
      ...databaseConfig,
    }),
    ConfigModule.forRoot({
      isGlobal: true,
      load: [config],
    }),
    UserModule,
    CategoryModule,
    ProductModule,
    BasketModule,
  ],
  controllers: [AppController],
  providers: [AppService, mailServiceProvider],
})
```

Для прикладу розглянемо модуль для роботи з товарами а саме `product.module.ts`.

```
@Module({
  imports: [HttpModule],
  providers: [ProductService],
  controllers: [ProductController],
  exports: [ProductService],
})
export class ProductModule {}
```

В файлі `product.module.ts` оголошується необхідний модуль за допомогою відповідного декоратора, і підключається необхідний сервіс `product.service.ts`, повний код файлу знаходиться в додатку А

В цьому файлі описані всі необхідні функції для роботи з товарами а саме `getProduct` та `getProduct`. Функція `getProduct` використовується для отримання інформації про товар за його `id` при переході на відповідну сторінку на клієнтській частині. `getProducts` використовується для отримання списку всіх товарів. Також описана логіка для фільтрації товарів за категорією, ціною, назвою та реалізовано сортування та пагінацію.

Для створення кінцевої точки було створено клас з потрібним декоратором,.

```
@ApiTags('Product')
@Controller('product')
export class ProductController {
    constructor(private readonly productService: ProductService) {}

    @ApiOperation()
    @Get('getProduct')
    getSubcategory(@Query() { productId }: GetProductDto) {
        return this.productService.getProduct(productId);
    }

    @ApiOperation()
    @Get('getProducts')
    getSubcategories(@Query() queryData: GetProductsDto) {
        return this.productService.getProducts(queryData);
    }
}
```

В цьому файлі описуються дві кінцеві точки, getProduct для отримання одного товару, та getProducts, для отримання списку товарів відповідно.

З боку клієнтської частини додатку було використано вебфреймворк NextJS 14, який створюється за допомогою інтерфейсу командного рядка. Вхідною точкою в додаток є папка app. Проект використовує AppRouter що означає що структура папок та файлів описує навігацію та структуру всього додатку. В каталозі з цим файлом присутній файл middleware.ts. Він відповідає за захист сторінок «ConfirmSignUp» і «Reset Password» від несанкціонованого входу.

```
export function middleware(request: NextRequest) {
    if (
        request.nextUrl.pathname === "/reset-password" ||
        request.nextUrl.pathname === "/confirm"
    ) {
        const token = request.nextUrl.searchParams.get("token");
        if (!token) return NextResponse.redirect(new URL("/catalog", request.url));
        return;
    }

    if (request.nextUrl.pathname === "/")
        return NextResponse.redirect(new URL("/catalog", request.url));
}
```

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						41

```
export const config = {
  matcher: [
    "/",
    "/reset-password",
    "/confirm",
  ],
};
```

Об'єкт config описує шляхи до сторінок на яких має спрацьовувати перевірка.

Авторизація користувача, а саме збереження токена в cookie реалізовано за допомогою «server action»[25]. Server action це функція яка відпрацьовує лише на серверній частині але її можна передати в клієнтський компонент через props. Таким чином вся робота що стосується авторизації користувача відбувається на сервері що суттєво підвищує безпеку і захищеність додатку. Нижче наведено код файлу layout.tsx який є вхідним файлом в додаток і в якому відбувається початкова перевірка стану авторизації, повний код файлу знаходиться в додатку А.

```
export async function updateCookie(token?: string) {
  "use server";
  cookies().set("vet-store-token", `${token}`, {
    httpOnly: true,
    path: "/",
  });
}
export default async function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  const token = cookies().get("vet-store-token");
  let isValid = false;
  try {
    await api.get("/user/me", {
      headers: { Authorization: `Bearer ${token?.value}` },
    });
    isValid = true;
  } catch (error) {
    isValid = false;
  }
  return ...
}
```

Після успішного логіну для оновлення токена викликається вище

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						42

описаний «Server action», і в cookie записується новий токен. Для того щоб викликати потрібну функцію її потрібно передати із серверного компонента в клієнтський, це відбувається в сторінці login.

```
async function Login() {
  await LogInGuard(cookies().get("vet-store-token")?.value);

  return (
    <div className="animate-show flex h-full w-full items-center justify-center py-6">
      <LoginForm updateCookie={updateCookie} />
    </div>
  );
}
```

Після чого ця функція викликається в потрібний момент після успішної авторизації.

```
function LoginForm({ updateCookie }: IProps) {
  const onSubmit = async (data: Schema) => {
    try {
      const { data: res } = await login(data);
      await updateCookie(res.access_token);
    } catch (error) {
      setError("root", {
        type: "custom",
        message: "Електронна пошта або пароль не вірні, спробуйте ще раз.",
      });
    }
  };
  ...
}
```

Всі сторінки додатку серверні, це означає що перед тим як відправитись користувачу, вони проходять ініціалізацію і будуються на сервері що дуже сильно спрощує реалізацію перевірки стану авторизації користувача. Для цього було створено функцію LogInGuard яка перевіряє чи токен дійсний шляхом отримання інформації про користувача. Помилка у відповіді означає що користувач не авторизований і його потрібно перенаправити на сторінку Login.

```
export async function LogInGuard(token?: string) {
  let res = undefined;
  try {
    const d = await api.get("/user/me", {
      headers: { Authorization: `Bearer ${token}` },
    });
    if (d.data) res = d.data;
  }
}
```

```

    } catch (error) {}
    if (res) redirect(routes.catalog);
  }

```

Приклад використання цієї функції можна побачити на сторінці оформлення замовлення.

```

async function Checkout() {
  await LogInChecker(cookies().get("vet-store-token")?.value);

  return (
    <div className="animate-show flex w-full flex-col gap-6 px-4 pb-4 ">
      ...

```

Для роботи чату підтримки було підключено openai. Для цього на стороні клієнту, в папку app було додано папку api. Особливість такого підходу в тому що те що знаходиться в папці api буде відпрацьовувати виключно на сервері, що дозволяє створювати та викликати кінцеві точки аналогічно як при використанні NestJS. Було створено окрему кінцеву точку /support в якому реалізована інтеграція з openai (повний код файлу додаток А).

```

const openai = new OpenAI({
  apiKey: process.env.CHAT_API_KEY,
});

export async function POST(req: NextRequest) {
  const body = (await req.json()) as { question: string };

  const completion = await openai.chat.completions.create({

  return NextResponse.json(
    { answer: completion.choices[0].message.content },
    { status: 200 },
  );
}

```

На клієнтській частині цей запит використовується аналогічно іншим.

```

export const getChatResponse = (question: string) =>
  axios.post<{ answer: string }>(Endpoints.get_chat_message, { question });

const onSubmit = async (data: Schema) => {
  setLoading(true);
  const res = await getChatResponse(data.question);
  ...
};

```

Для роботи з товарами та категоріями було використано бібліотеку «Tanstack query». Вона надає дуже великий функціонал для роботи з запитами, автоматичне збереження даних для пришвидшення наступних завантажень і тд. Для її використання було спочатку було створено клієнтський компонент провайдер оскільки даний код може виконуватись лише на клієнтській частині

```
const queryClient = new QueryClient();
```

```
function QueryClientWrapper({ children }: TProps) {  
  return (  
    <QueryClientProvider client={queryClient}>{children}</QueryClientProvider>  
  );  
}  
export default QueryClientWrapper;
```

Після чого було створено обгортку над основним хуком «useQuery» для полегшення його використання в майбутньому, повний код файлу знаходиться в додатку А.

```
function useBaseQuery<T>(queryKey, request, options): UseQueryResult<T | undefined,  
unknown> {  
  return useQuery({  
    queryKey,  
    queryFn: async function ({ signal }) {  
      return await request(signal).data;  
    },  
    ...options,  
  });  
}  
export default useBaseQuery;
```

Далі для отримання продуктів було створено окремий хук.

```
export const useRecomendedProductsQuery = (params: IProductParams) => {  
  return useBaseQuery(["products", params], () => getAllProducts(params), {});  
};
```

Після чого його дуже зручно можна використовувати потрібних компонентах.

```
function PageContent({ product }: { product: IProduct }) {  
  const productsQuery = useRecomendedProductsQuery({  
  });  
  const recomended = productsQuery.data ?? [];  
  return (  
    ...  
    <div className="flex flex-wrap items-center justify-center gap-8 pb-24">  
      {recomended.map((item) => (  
        <ProductCard key={item.id} product={item} />  
      )}
```

```

    ))}
  </div>
);

```

Відповідно для всіх необхідних кінцевих точок було створено власні хуки що дуже сильно спрощує код, підвищує надійність та читабельність.

Вміст корзини зберігається в local storage, і якщо користувач авторизований в базі даних. Для реалізації цього функціоналу було використано бібліотеку zustand.

Це забезпечує глобальний сховище для збереження даних яке доступне з будь якого компоненту додатку. Для збереження його в «local storage» було використано вбудовану в zustand функцію. Також при авторизації користувача відбувається автоматична синхронізація з тими даними що були збережені в базі даних раніше

```

function UserMenu({ logOut, token }: IProps) {
  const syncCarts = useCartStore((store) => store.syncCarts);

  useEffect(() => {
    syncCarts(!token);
  }, [syncCarts, token]);
  ...

```

Якщо такі є вони записуються в local storage, якщо інформація в базі даних відсутня і в до «local storage» раніше було додано товари то ці дані буде збережено до бази даних, повний код файлу знаходиться в додатку А.

```

syncCarts: async (login) => {
  set((state) => ({ items: state.items, isLogIn: login }));
  if (!login) {
    return;
  }
  const products = get().items;
  try {
    const { data } = await getUserCart();
    if (products.length > 0 && data.basketItems.length === 0) {
    }
    if (data.basketItems.length > 0) {
    }
  } catch {
    set((state) => ({ ...state }));
  }
},

```

Для реалізації функції «Завантажити ще» на сторінці товарів було використано окремий хук `useInfiniteQuery`. Він особливий тим що при зміні ключа, до попередньо завантажених даних додаються нові щойно завантажені дані що дає змогу зробити нескінченне завантаження товарів що суттєво покращує користувацький досвіт. На основі цього хука було написано власний хук `useProductsQuery` що сильно спрощує подальше використання цієї функції.

```
export const useProductsQuery = (
  params: Omit<IProductParams, "skip" | "take">,
  perPage: number,
) => {
  return useInfiniteQuery({
    queryKey: ["products", params, perPage],
    queryFn: async (p) => {
      const d = await getAllProducts(

      return ...
    },
    initialPageParam: 0,
    getNextPageParam: (lastPage) => lastPage.next,
    getPreviousPageParam: (firstPage) => firstPage.prev,
  });
};
```

В потрібному компоненті потрібно лише викликати потрібний хук і відмалювати результат.

```
const productsQuery = useProductsQuery({});
const pages = productsQuery.data?.pages ?? [];
...
{pages.map((page, index) => (
  <Fragment key={index}>
    {page.data?.map((item) => (
      <ProductCard key={item.id} product={item} />
    ))}
  </Fragment>
))}
```

Висновки до другого розділу:

У процесі проектування додатку для продажу ветеринарних товарів були детально проаналізовані варіанти використання програмного продукту згідно з вимогами програмного забезпечення. В результаті цього аналізу була спроектована структура системи, яка включає взаємодію між основними компонентами додатку.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						47

Було розроблено базу даних, яка забезпечує ефективне зберігання та обробку інформації про товари, користувачів, замовлення та інші сутності системи. Створені таблиці та діаграма бази даних дозволяють візуалізувати структуру даних та їх взаємозв'язки.

Під час проектування та реалізації алгоритмів роботи додатку було розроблено основні функціональні процеси, включаючи обробку замовлень, управління користувачами, пошук товарів та їх фільтрацію. Описані алгоритми враховують логіку роботи додатку та забезпечують коректне виконання усіх бізнес-процесів.

Реалізація функціоналу додатку включала створення інтерфейсу користувача та серверної частини, забезпечення безпеки даних та інтеграцію з платіжними системами. В результаті було створено повнофункціональний додаток, який відповідає визначеним вимогам та забезпечує зручність використання для кінцевих користувачів.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ ДЛЯ ПРОДАЖУ ВЕТЕРИНАРНИХ ТОВАРІВ

3.1 Порядок встановлення та налаштування параметрів додатку

Додаток для продажу ветеринарних товарів складається з кількох компонентів, кожен з яких потребує окремого розгортання та налаштування. Для відображення цих компонентів та візуалізації процесу їх розгортання було створено відповідну діаграму.

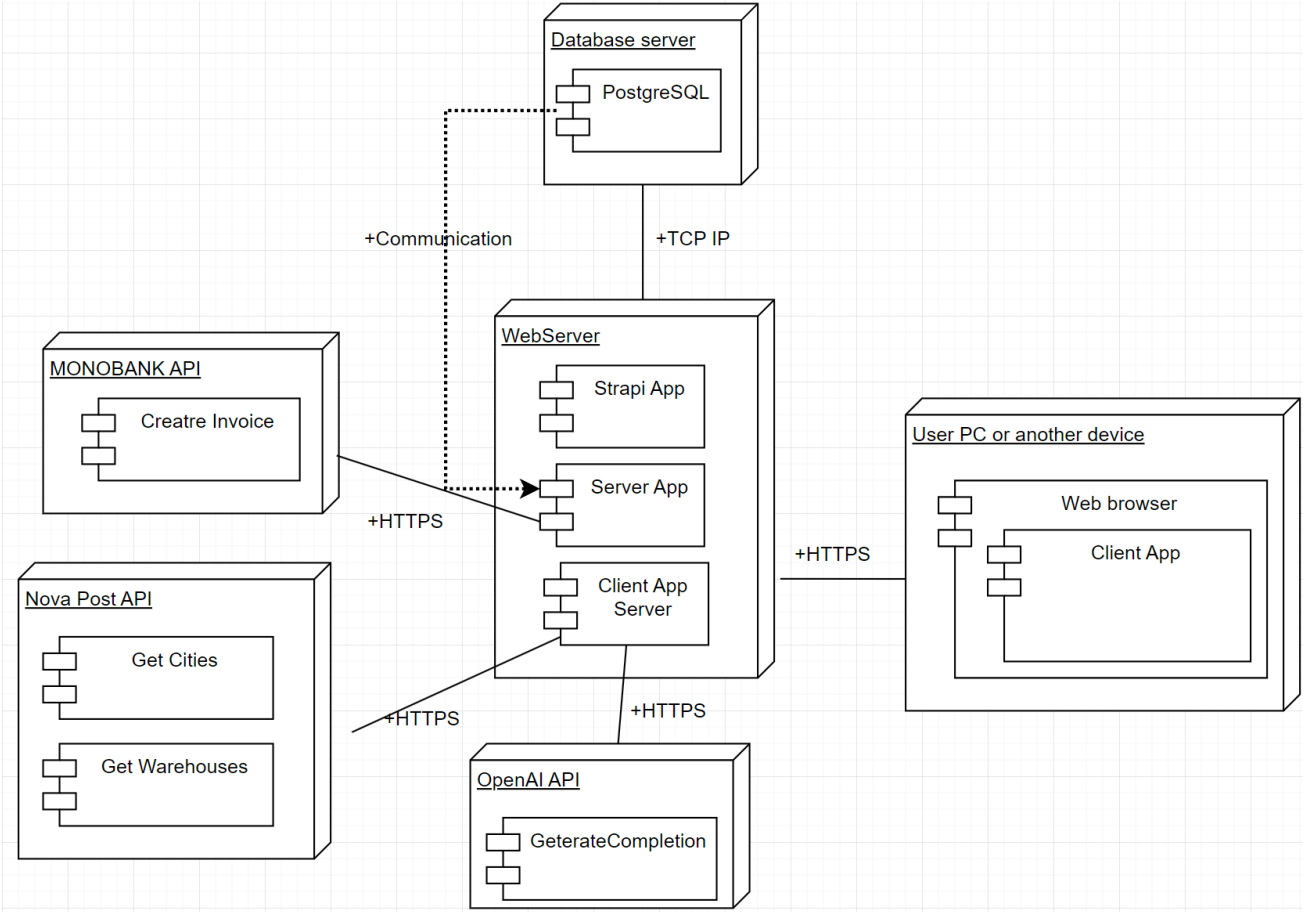


Рисунок 3.1 – Діаграма розгортання

На діаграмі розгортання зображена система веб-додатку для продажу ветеринарних товарів, яка складається з шести основних вузлів: серверу бази даних, веб-серверу, зовнішніх API, та комп'ютера або іншого пристрою користувача. Кожен компонент має свою роль і специфічні протоколи для взаємодії.

Сервер бази даних використовує PostgreSQL для зберігання даних. Він забезпечує зберігання і обробку інформації про товари, користувачів, замовлення та інші сутності системи. Зв'язок між сервером бази даних та веб-сервером здійснюється через протокол TCP/IP, що забезпечує надійну передачу даних без втрат.

Веб-сервер виконує ключову роль у функціонуванні додатку. На ньому розгорнуті три основні компоненти: Strapi App, що використовується для адміністративної частини, Server App на NestJS, який відповідає за логіку серверної частини, та Client App на NextJS, який відповідає за функціонал клієнтської частини. Взаємодія між цими компонентами та з іншими сервісами відбувається через протокол HTTPS, що забезпечує захищену передачу даних.

Зовнішні API включають MONOBANK API для створення обробки та валідації інтернет оплат, Nova Post API для отримання даних про міста та пункти видачі, та OpenAI API для генерації відповідей на питання користувачів. Всі ці API інтегровані з системою через протокол HTTPS, що гарантує безпечний обмін даними.

Комп'ютер або інший пристрій користувача забезпечує інтерфейс для взаємодії з додатком через веб-браузер. Клієнтський додаток працює безпосередньо в браузері користувача, а зв'язок з веб-сервером здійснюється через протокол HTTPS, що гарантує безпеку даних під час їх передачі.

Для належного функціонування додатку необхідно встановити PostgreSQL на сервері бази даних, створити необхідні бази даних та налаштувати підключення до них. На веб-сервері потрібно встановити операційну систему Linux. Наступним кроком потрібно встановити Docker оскільки кожна частина вебдодатку повинна розгортатись в окремому контейнері для запобігання конфліктів. Наступним кроком буде встановлення NodeJS. Після виконання цих кроків потрібно розгорнути Strapi для адміністративної частини, Server App на NestJS, та Client App на NextJS, а також налаштувати HTTPS для забезпечення безпечної взаємодії між компонентами, та прив'язати до адреси сервера відповідне доменне ім'я.

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						50

Для правильної роботи сервера необхідно мати версію Node.JS версії 20 або вище. Також переглянемо файл налаштувань налаштування середовищ для серверної та клієнтської частин.

Спочатку переглянемо файл env серверної частини.

1. PORT = Вказує порт, на якому буде працювати веб-сервер.
2. DB_HOST = Вказує адресу бази даних PostgreSQL.
3. DB_PORT = Вказує порт, на якому база даних PostgreSQL прослуховує з'єднання. Значення 5432 є стандартним портом для PostgreSQL.
4. DB_NAME = Ім'я бази даних, яку використовуватиме додаток.
5. DB_USERNAME = Ім'я користувача для підключення до бази даних.
6. DB_PASSWORD = Пароль для підключення до бази даних.
7. DB_LOGGING = Вказує, чи буде логування запитів до бази даних увімкнено. Може мати значення true або false.
8. STRAPI_URL = URL-адреса для підключення до Strapi, адміністративної частини додатку.
9. FRONTEND_SUCCESS_LINK = URL-адреса, на яку буде перенаправлено користувача для підтвердження електронної адреси

Також переглянемо файл env клієнтської частини.

1. NEXT_PUBLIC_NP_API_KEY = Секретний ключ для доступу до Nova Post API.
2. NEXT_PUBLIC_NP_API_URL = URL-адреса для підключення до Nova Post API
3. NEXT_PUBLIC_API_URL = URL-адреса на якій працює серверна частина
4. NEXT_PUBLIC_CHAT_API_KEY = Секретний ключ для доступу до OpenAI.

Зовнішні API, такі як MONOBANK, Nova Post та OpenAI, повинні бути налаштовані для обміну даними через захищені канали. Клієнтські пристрої повинні мати доступ до мережі інтернет та до сучасних веб-браузерів для роботи з додатком.

3.2 Структура інтерфейсу веб-додатку та порядок роботи

Для використання додатку користувачам потрібний пристрій зі встановленим інтернет браузером. Завдяки адаптивності інтерфейсу створеного вебдодатку, користувачі можуть зручно користуватися веб сайтом як на мобільних пристроях так і на персональних комп'ютерах.

Переш що побачить користувач це каталог товарів. У шапці додатку присутні кнопки для навігації між сторінками, логотип, при натисканні на який, користувача направляє на головну сторінку сайту, кнопка для відкриття корзини та, в залежності від стану авторизації користувача, кнопка яка веде на сторінку авторизації у випадку якщо користувач не авторизований або кнопка для відкриття меню користувача у випадку якщо користувач авторизований.

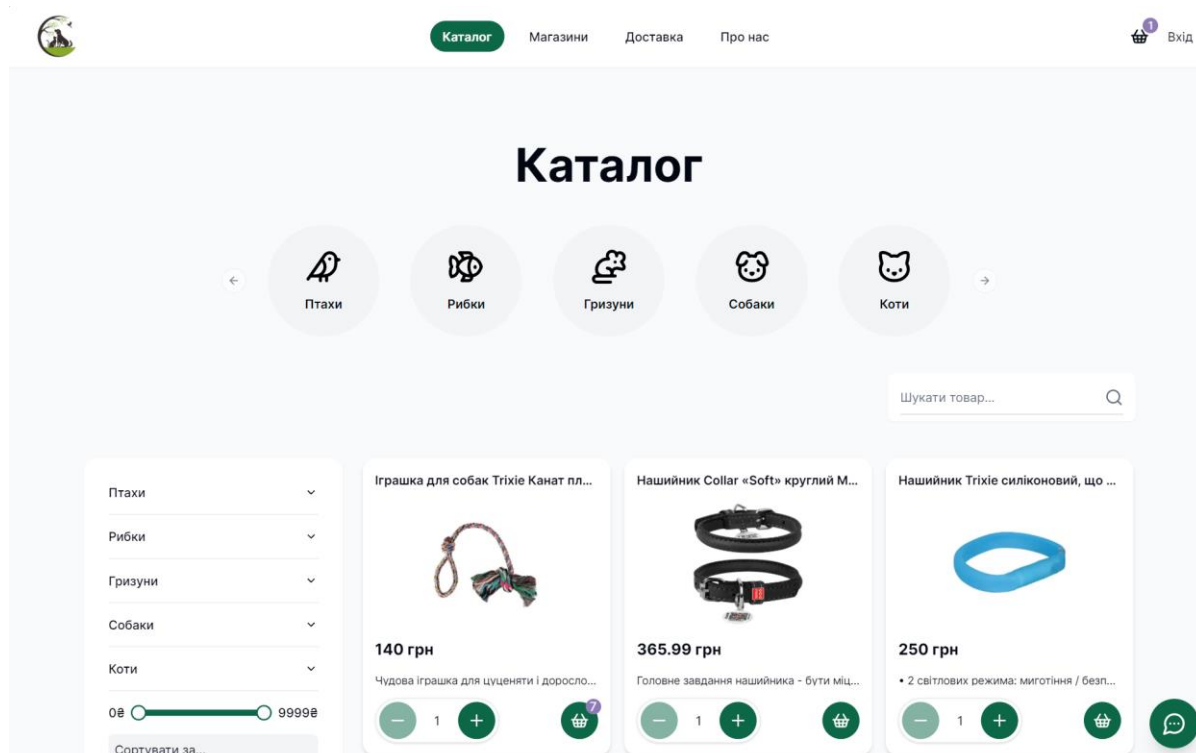


Рисунок 3.2 – Головна сторінка

По замовчуванню користувачу не доступний особистий кабінет. Він доступний лише для авторизованих користувачів. Для авторизації потрібно перейти на сторінку Login та ввести логін і пароль.

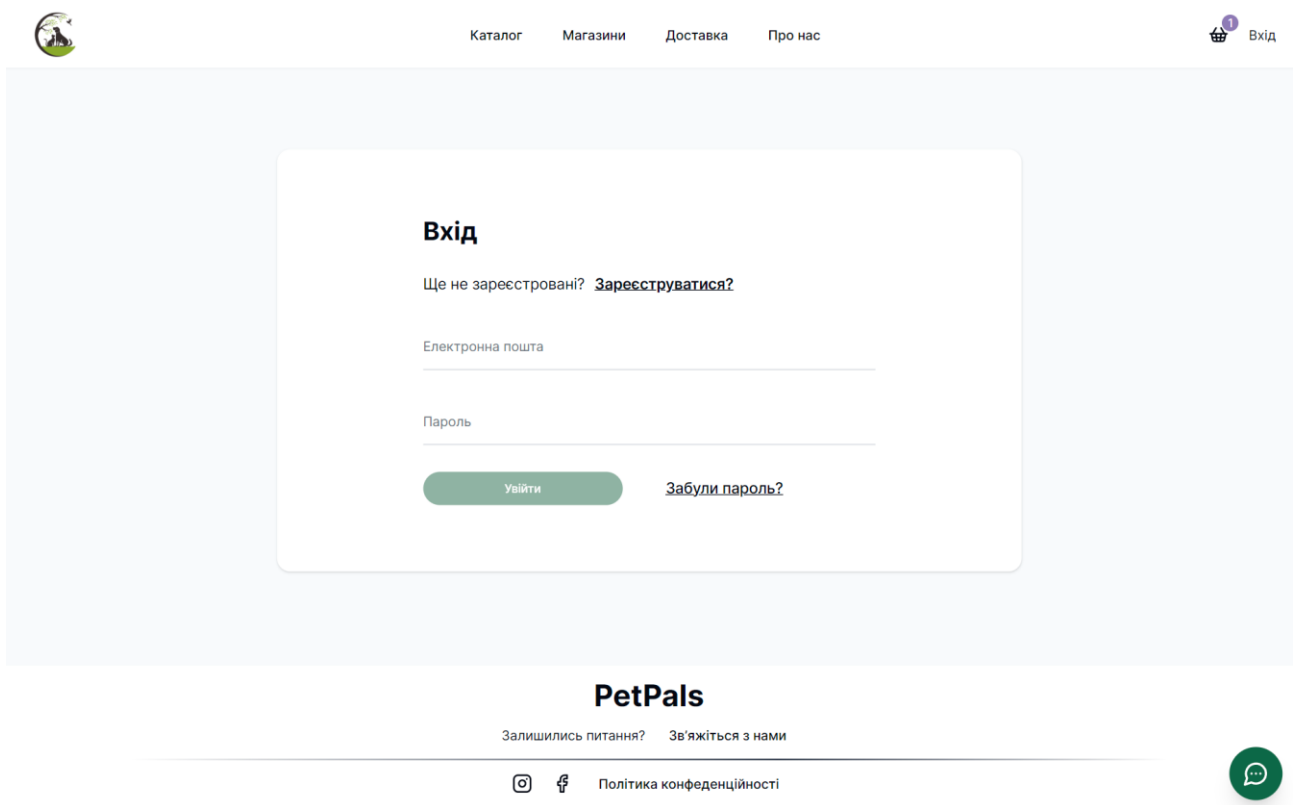


Рисунок 3.3 – Сторінка авторизації користувача.

Після успішної авторизації вигляд шапки сайту дещо змінюється і замість посилання на сторінку входу відображається кнопка яка відкриває меню користувача. Натиснувши на цю кнопку для користувача відкривається меню в якому знаходяться посилання на сторінку з особистою інформацією та на сторінку зі списком замовлень користувача

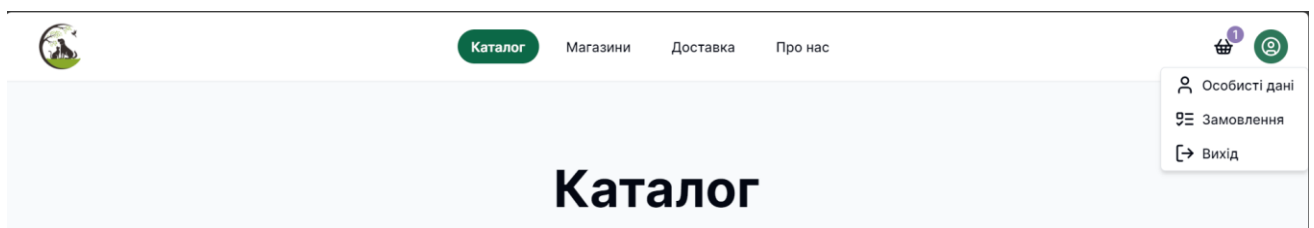


Рисунок 3.4 – Шапка сайту після успішної авторизації.

Якщо користувач вже був авторизований то при вході на сайт одразу буде відображатись другий варіант шапки,

На головній сторінці сайту користувач має змогу переглядати список товарів. Є можливість пошуку товару за назвою, сортування за ціною, фільтрація за категоріями і підкатегоріями, можливість встановлення мінімальної і

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						53

максимальної цін. Також користувач може завантажити більше товарів натиснувши на кнопку «Завантажити ще»

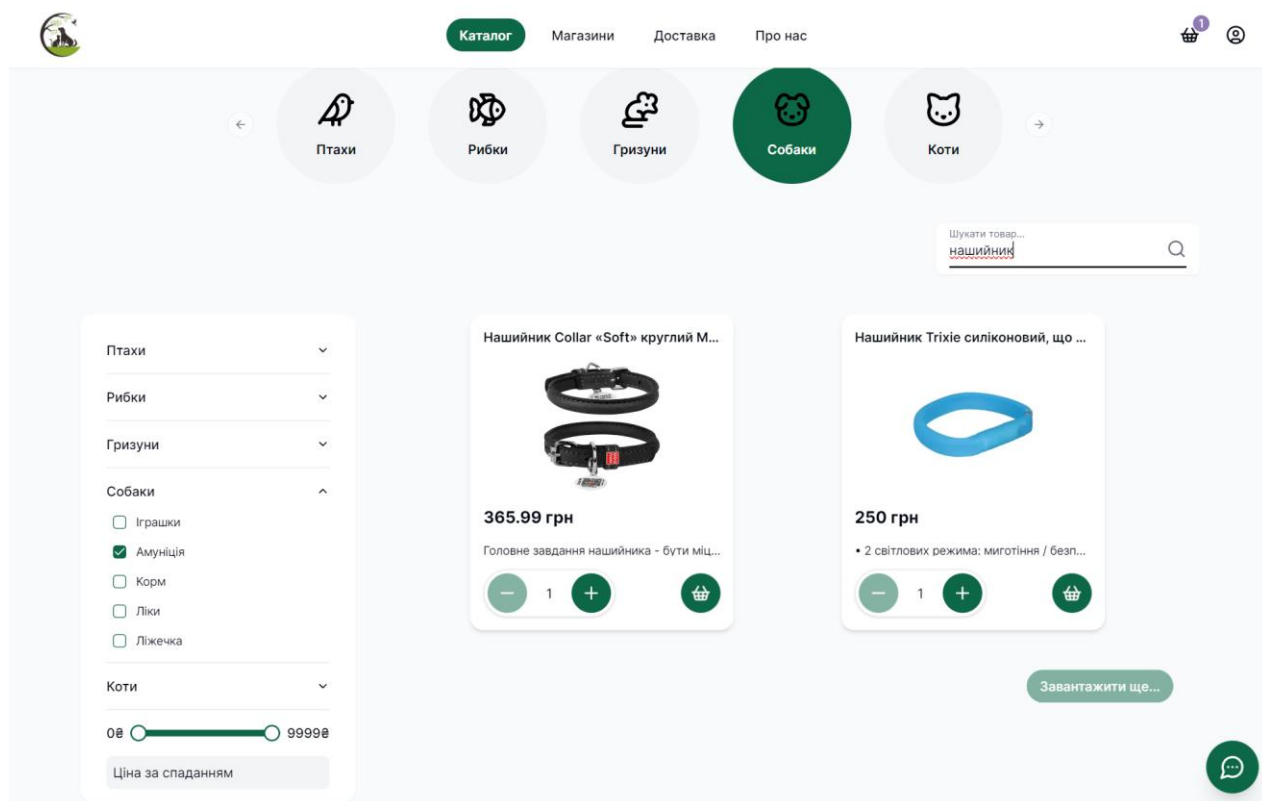


Рисунок 3.5 – Список продуктів при обраних фільтрах.

Користувач може обрати кількість товарів та додати товар до кошика. Якщо товар вже додано до кошику то обрано користувачем кількість буде додана до кількості вже доданих раніше товарів. В шапці сайту поруч з кнопкою для відкриття кошика відображується кількість доданих позицій.



Рисунок 3.6 –Вигляд шапки сайту у випадку якщо до кошика додано товари.

Також змінюється вигляд картки товару, а саме поруч з кнопкою «Додати до кошика» відображується скільки саме одиниць цього товару було додано до кошика.

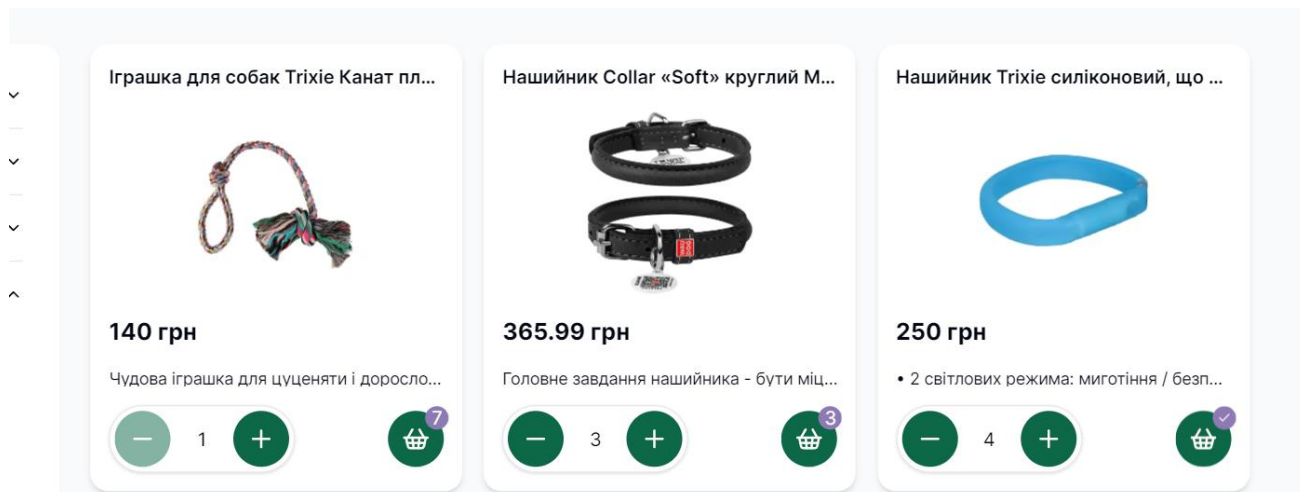


Рисунок 3.7 –Вигляд картки товару якщо цей товар додано до кошика.

При натисканні на карту товару відкривається сторінка з детальною інформацією про товар. На цій сторінці відображується повний опис товару, його категорія, бренд, і список рекомендованих товарів. Також є можливість додати цей товар до кошика, обравши його кількість і натиснувши кнопку «Купити».

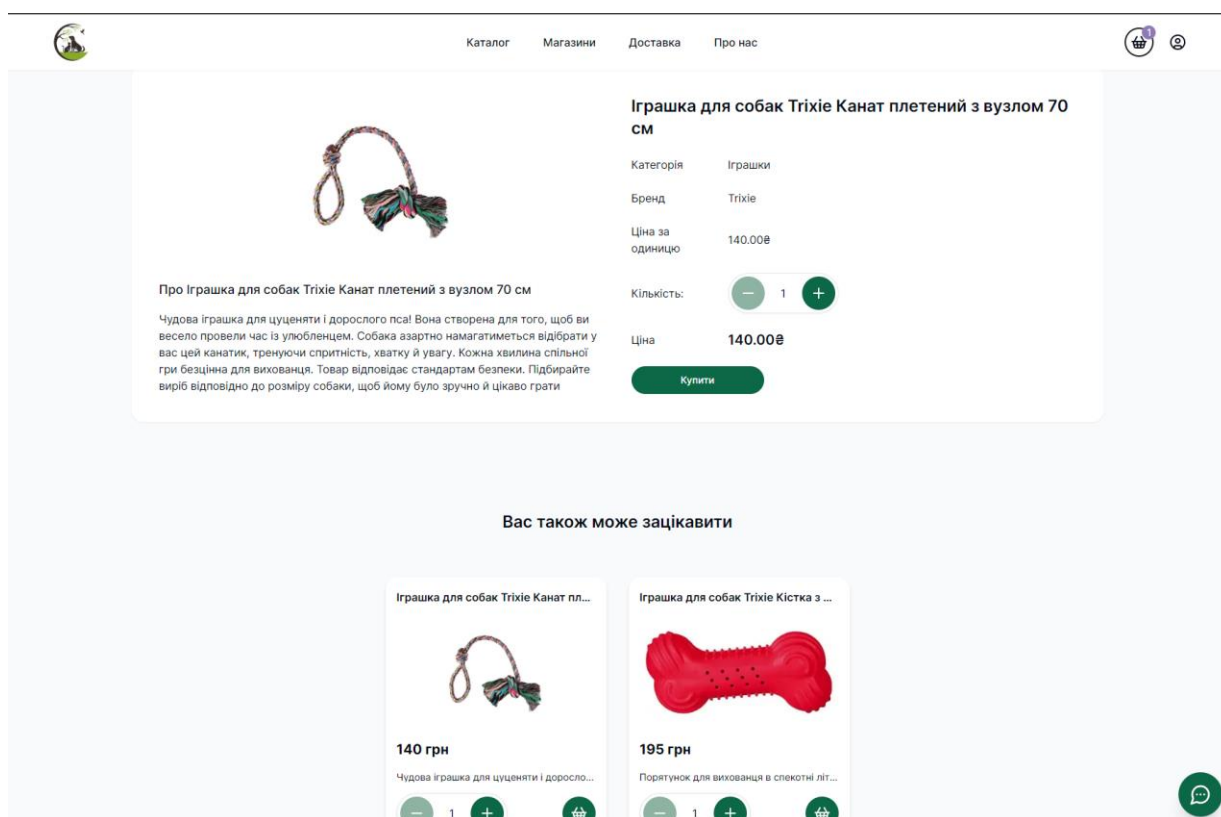


Рисунок 3.8 –Сторінка товару.

Користувач має змогу переглянути вміст кошика натиснувши на відповідну кнопку в шапці сайту. Після чого на екрані відкривається діалогове вікно в якому користувач може переглянути вміст кошика. Також користувач може змінювати кількість певного товару, видаляти товари або видалити одразу всі товари натиснувши на кнопку «Очистити кошик». Також можна переглянути загальну ціну всіх доданих товарів.

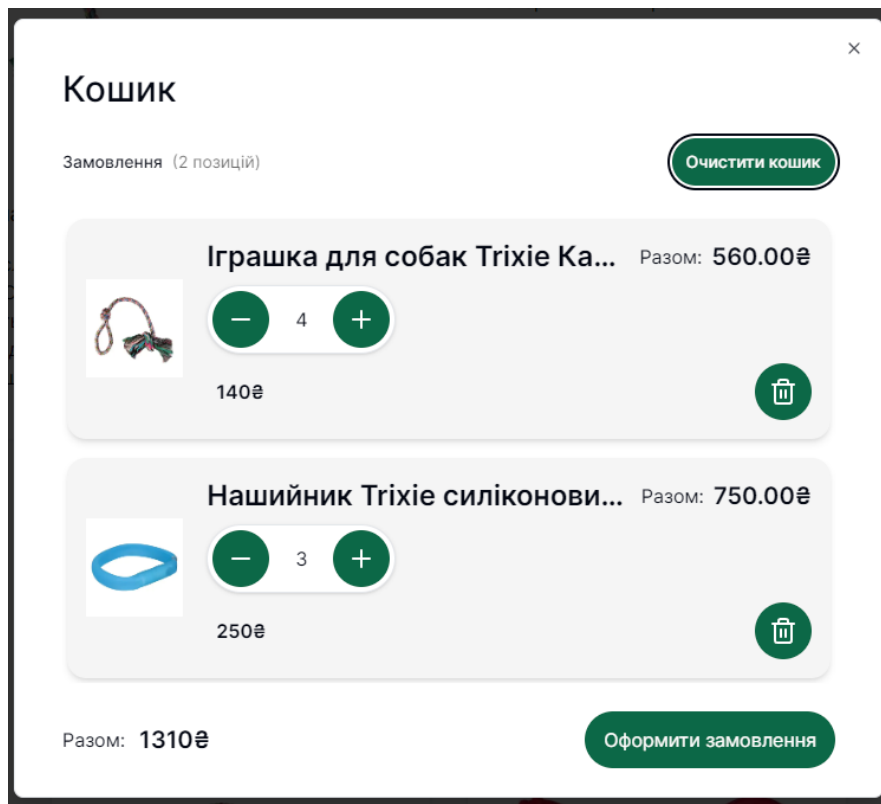


Рисунок 3.9 –Кошик користувача.

Авторизований користувач має змогу редагувати особисту інформацію перейшовши на відповідну сторінку.

При реєстрації ця інформація не є обов'язковою. У випадку якщо користувач вирішить надати цю інформацію, на етапі оформлення замовлення всі потрібні поля, а саме ім'я прізвище номер телефону будуть автоматично заповнені. Також в кінці знаходиться кнопка «Вихід» натиснувши на яку користувач виходить зі свого облікового запису, наприклад якщо користувачу необхідно змінити обліковий запис або якщо користувач не хоче щоб інші користувачі даного пристрою переглядали його особисту інформацію.

Для того щоб редагувати особисту інформацію користувач має перейти на відповідну сторінку. На цій сторінці він може переглянути інформацію про себе та своїх домашніх улюбленців, та за бажанням редагувати цю інформацію натиснувши на кнопку «Редагувати» у відповідному блоці.

Рисунок 3.10 – Особистий кабінет користувача.

Користувач має можливість переглядати та керувати своїми замовленнями. Є можливість фільтрувати замовлення за різними періодами. Можна обрати період за останні 30 днів, за останні 6 місяців і тд, або обрати конкретний день, щоб швидко знайти необхідну інформацію.

Для того щоб переглянути деталі замовлення потрібно натиснути на картку замовлення.

У випадку якщо у клієнта з’являється питання він може скористатися вбудованим чат-ботом.

Для того щоб відкрити чат підтримки цього необхідно натиснути відповідну кнопку в правому нижньому куту додатку.

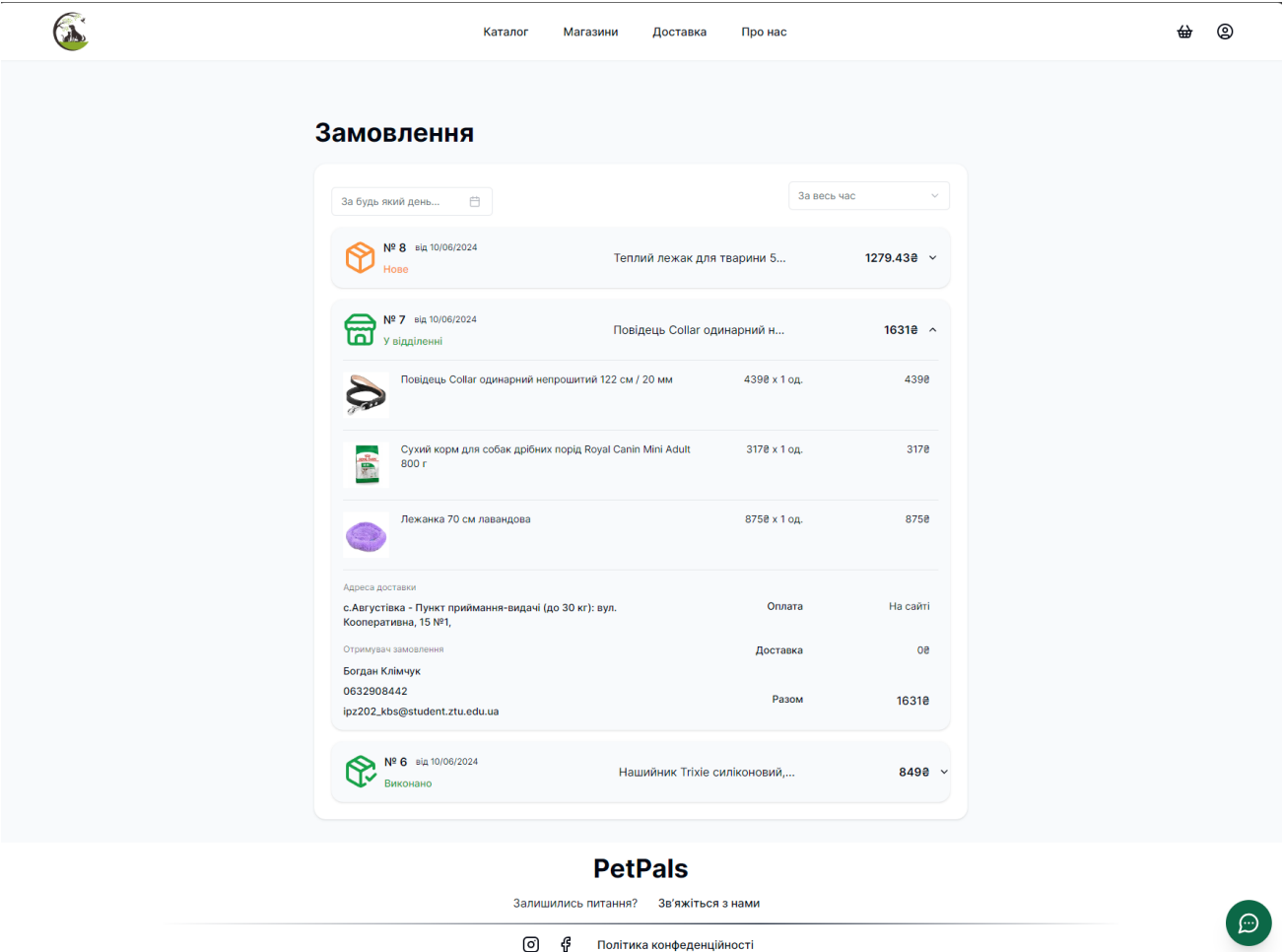


Рисунок 3.11 – Сторінка з замовленнями користувача.

У випадку якщо у клієнта з’являється питання він може скористатися вбудованим чат-ботом створеним з використанням OpenAI. Він використовує потужну нейронну мережу для генерації максимально чіткої відповіді на питання користувачів.

Для того щоб відкрити чат підтримки цього необхідно натиснути відповідну кнопку в правому нижньому куту додатку.

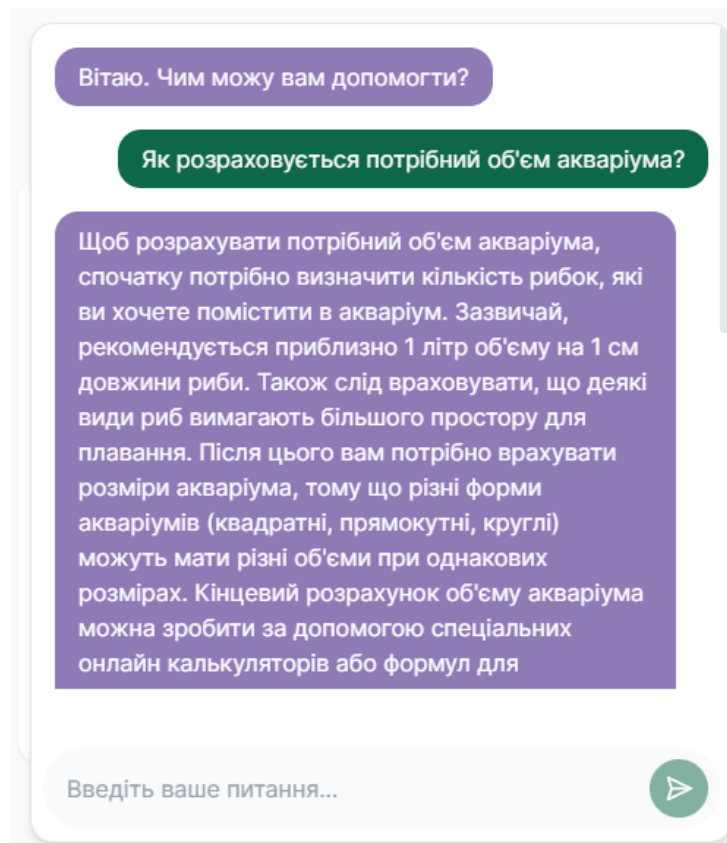


Рисунок 3.12 –Чат підтримки.

Також користувач може переглянути інформацію про доставку та оплату перейшовши на відповідну сторінки.

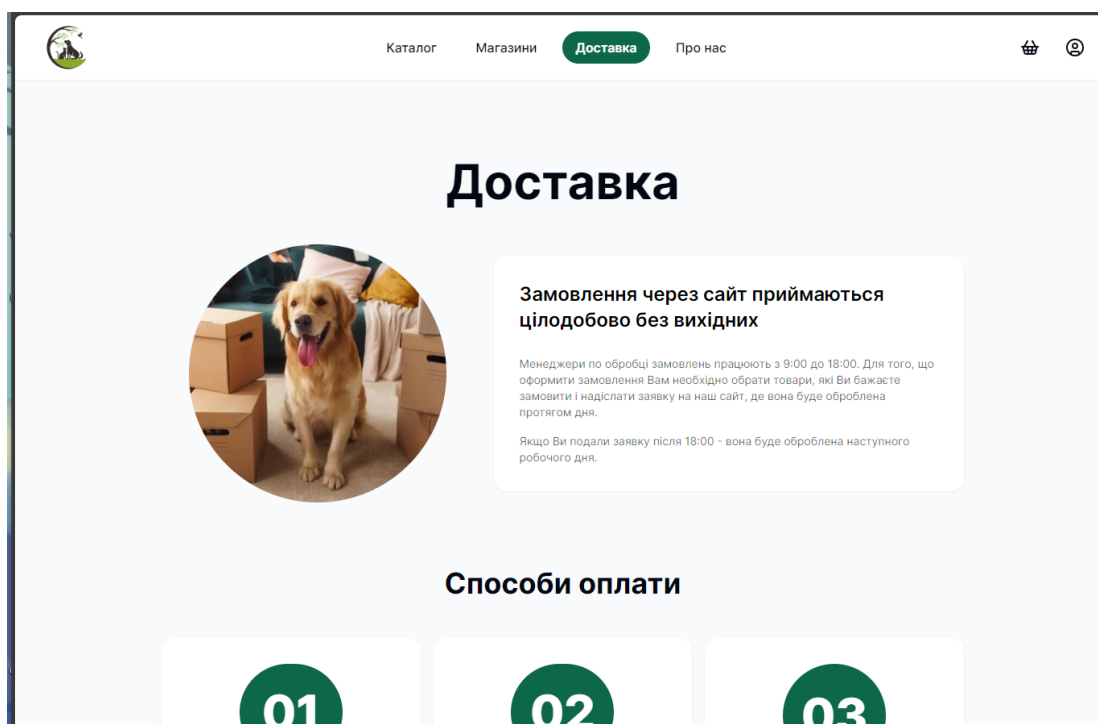


Рисунок 3.13 – Інтерфейс сторінки «доставка».

На цій сторінці можна переглянути інформацію про те в який час приймаються замовлення. Як Яким чином відбувається оплата, та вартість доставки при різних умовах.

На сторінці про нас знаходиться інформація про компанію та команду.

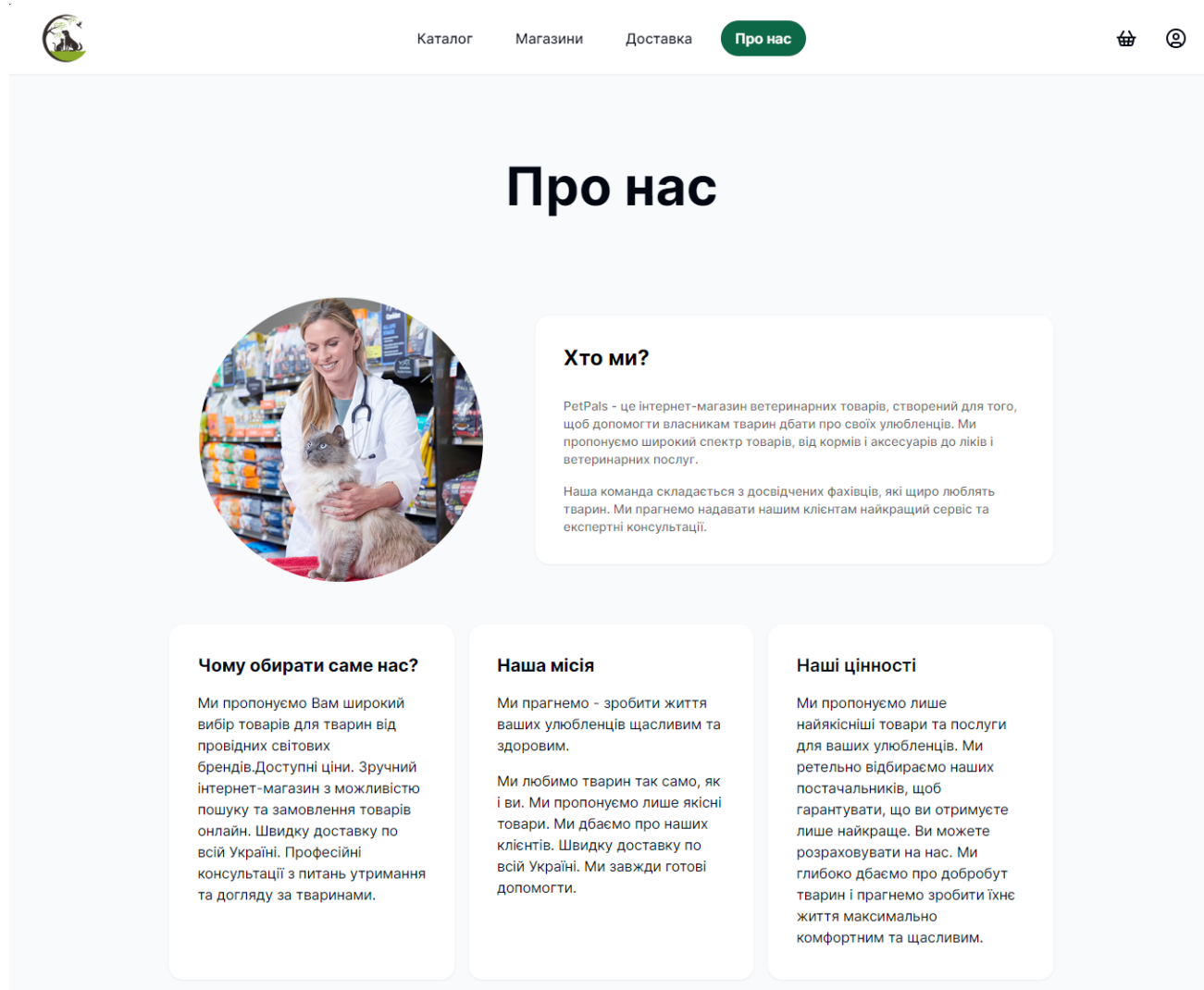


Рисунок 3.14 – Інтерфейс сторінки «Про нас».

3.3 Тестування роботи веб-додатку

За допомогою онлайн сервісу було проведено тестування вебдодатку з використанням GTMetrix[24].

GTMetrix є інструментом для перевірки продуктивності та швидкості завантаження вебсторінок. Він генерує звіти та аналізує продуктивність вебсайту за різними показниками, такими як час завантаження сторінки, розмір

сторінки, кількість запитів до сервера тощо. Після завершення тестування було отримано звіт, що містить різноманітну інформацію про продуктивність сайту (рис. 3.3.1).

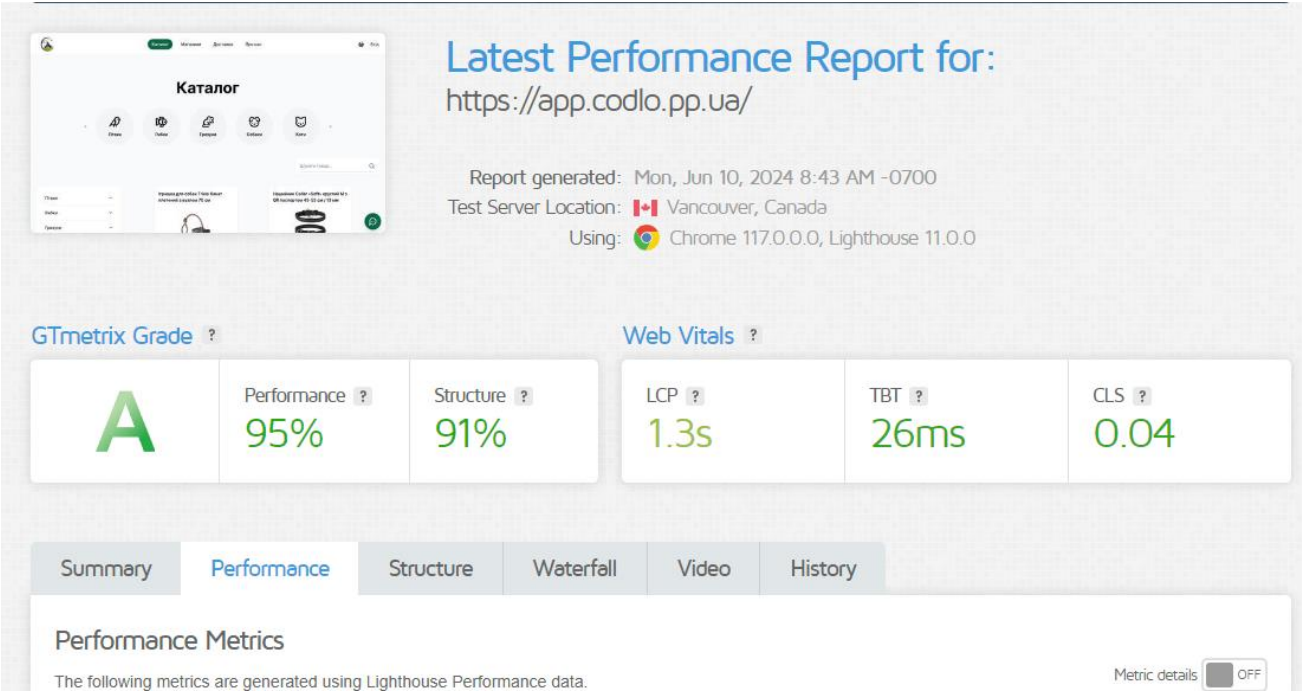


Рисунок 3.15– Тестування додатку за допомогою GTMetrix

Звіти GTMetrix включають такі дані, як загальний час завантаження сторінки, час завантаження вмісту, кількість запитів до сервера, розмір сторінки, швидкість завантаження на різних типах з'єднань тощо. Ця інформація допомагає зрозуміти, як створений сворений додаток для продажу ветеринарних товарів працює з точки зору продуктивності та швидкості, і дозволяє вжити заходів для покращення цих показників.

Веб-додаток отримав високу оцінку A в GTMetrix, що свідчить про відповідність сучасним нормам щодо продуктивності та швидкості завантаження. Це означає, що веб-додаток працює дійсно швидко і ефективно, що позитивно впливає на досвід користувачів, забезпечуючи їм зручний та швидкий інтерфейс.

GTMetrix також пропонує низку інструментів та функцій для оптимізації продуктивності вебсайту, зокрема можливість зберігати та порівнювати звіти,

слідкувати за показниками продуктивності сайту, отримувати сповіщення про проблеми продуктивності тощо.

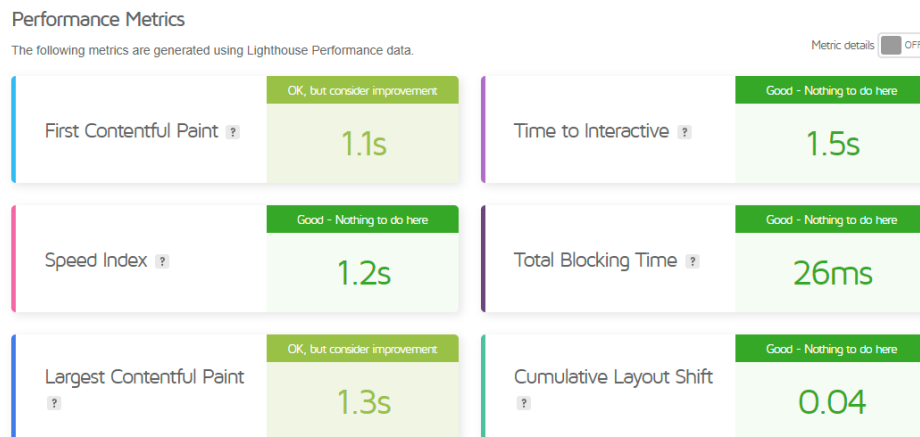


Рисунок 3.16 – Тестування швидкості додатку за допомогою GTMetrix

Переглянувши отримані данні можна зробити висновок що деякі частини додатку потребують оптимізації. А саме підвищити швидкість відтворення картинок та тексту. Також є можливість покращити швидкість завантаження картинок шляхом зменшення їх розміру.

Було проведено тестування інтерфейсу додатку на мобільну готовність оскільки більшість частіше користуються мобільним телефоном.

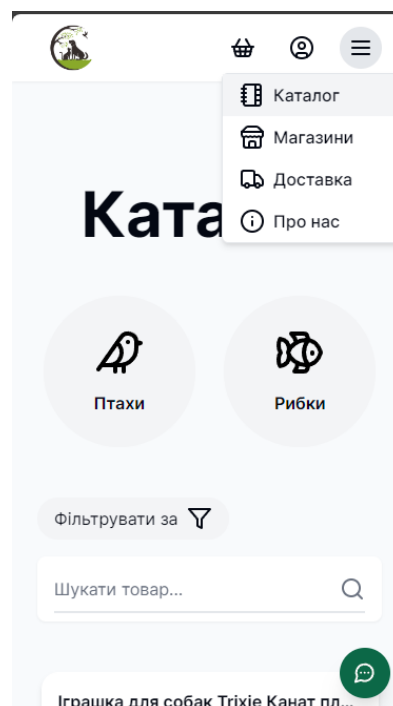


Рисунок 3.17 – Вигляд сторінки «Каталог» на мобільному пристрої

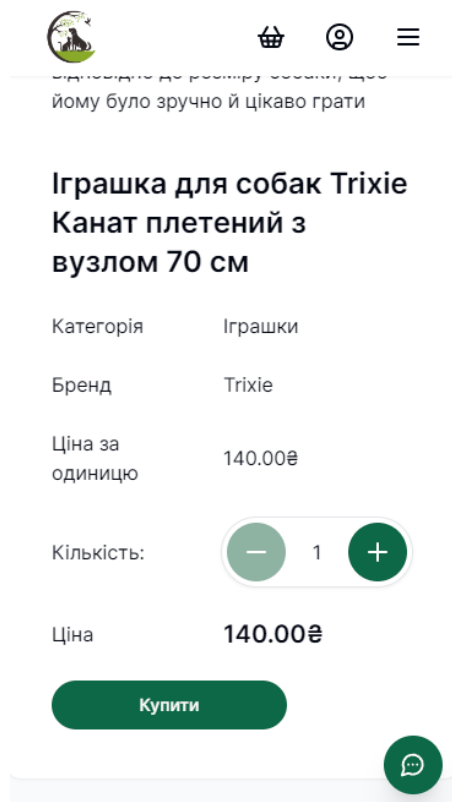


Рисунок 3.18 – Вигляд сторінки «Продукт» на мобільному пристрої

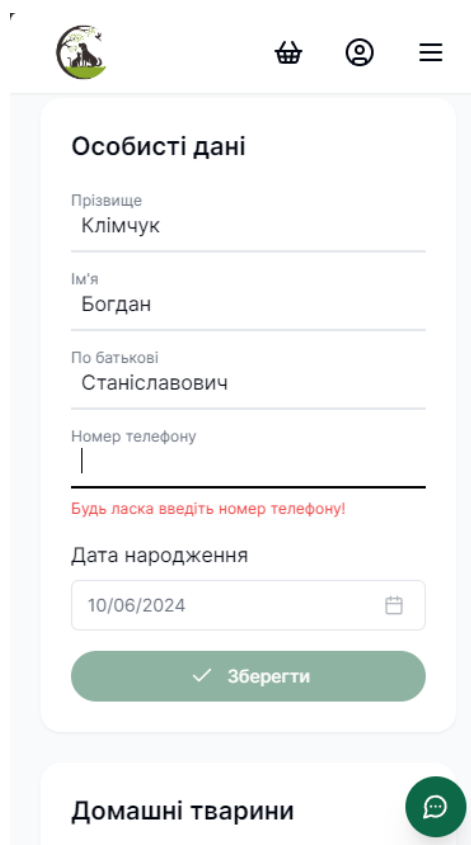


Рисунок 3.19 – Вигляд сторінки «Особисті дані» на мобільному пристрої

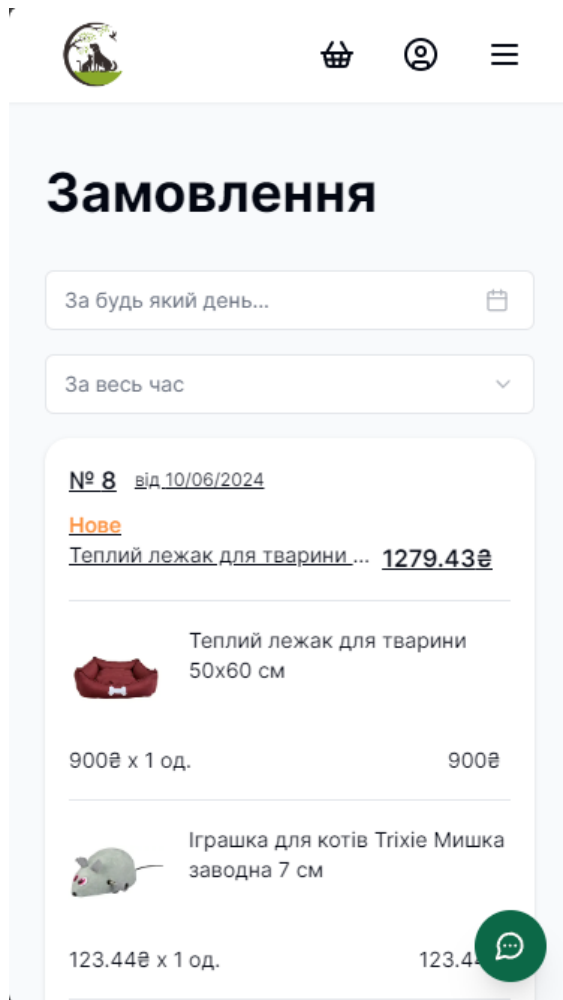


Рисунок 3.20 – Вигляд сторінки «Замовлення» на мобільному пристрої

Тестування вебдодатку на мобільних пристроях показало високий рівень адаптивності. Сайт чудово пристосовується до різних розмірів екранів, забезпечуючи зручний та інтуїтивно зрозумілий інтерфейс для користувачів мобільних пристроїв. Усі функції та елементи дизайну зберігають свою функціональність і ефективність, що гарантує комфортне використання додатку незалежно від типу та розміру мобільного пристрою.

Висновки до третього розділу

У цьому розділі було детально розглянуто порядок роботи з веб-додатком для продажу ветеринарних товарів. Було надано покрокову інструкцію щодо

встановлення та налаштування параметрів додатку, що дозволяє користувачам легко інтегрувати його у свої системи.

Також було описано структуру інтерфейсу веб-додатку та порядок роботи з ним. Розглянуто всі функціональні можливості додатку, що забезпечують зручну навігацію та ефективне користування для різних типів користувачів, включаючи адміністраторів і звичайних користувачів.

Крім того, проведено тестування роботи веб-додатку, що підтвердило його стабільність та високу продуктивність. Це гарантує, що додаток здатний ефективно виконувати свої функції та забезпечувати якісний сервіс для користувачів.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було спроектовано та розроблено вебдодаток для продажу ветеринарних товарів. Проведено детальний аналіз проблематики, методів і засобів вирішення задачі, а також порівняння аналогів програмного продукту. На основі цього аналізу визначено основні вимоги до вебдодатку та обґрунтовано вибір інструментальних засобів і апаратного забезпечення.

Проектування вебдодатку включало визначення варіантів використання та структури системи, розробку бази даних і реалізацію алгоритмів роботи додатку. Особлива увага приділена безпеці та захисту даних, що є критично важливим для електронної комерції. Було створено функціонал, що забезпечує зручність використання як для адміністраторів, так і для звичайних користувачів.

У розділі, присвяченому інтерфейсу та порядку роботи з вебдодатком, надано покрокову інструкцію з встановлення та налаштування параметрів додатку. Описано структуру інтерфейсу та порядок роботи з ним, що дозволяє користувачам ефективно орієнтуватися в системі та використовувати всі її можливості. Проведено тестування вебдодатку, яке підтвердило його стабільність та високу продуктивність на мобільних пристроях, що гарантує якісний сервіс для кінцевих користувачів.

Загалом, розроблений вебдодаток для продажу ветеринарних товарів відповідає встановленим вимогам, забезпечує необхідний функціонал та демонструє високу адаптивність і продуктивність, що дозволяє ефективно вирішувати задачі, пов'язані з продажем ветеринарних товарів онлайн.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Pethouse [Електронний ресурс] - Режим доступу до ресурсу: <https://pethouse.ua/>
2. E-zoo [Електронний ресурс] - Режим доступу до ресурсу: <https://e-zoo.com.ua/>
3. Zooboom [Електронний ресурс] - Режим доступу до ресурсу: <https://zooboom.od.ua/>
4. NestJS [Електронний ресурс] - Режим доступу до ресурсу: <https://nextjs.org>
5. NextJS [Електронний ресурс] - Режим доступу до ресурсу: <https://nextjs.org>
6. NextJS SSR [Електронний ресурс] - Режим доступу до ресурсу: <https://nextjs.org/docs/pages/building-your-application/rendering/server-side-rendering>
7. NextJS SSG [Електронний ресурс] - Режим доступу до ресурсу: <https://nextjs.org/docs/pages/building-your-application/rendering/static-site-generation>
8. NextJS CSR [Електронний ресурс] - Режим доступу до ресурсу: <https://nextjs.org/docs/pages/building-your-application/rendering/client-side-rendering>
9. TypeScript [Електронний ресурс] - Режим доступу до ресурсу: <https://www.typescriptlang.org/>
10. Date-fns [Електронний ресурс] - Режим доступу до ресурсу: <https://date-fns.org/>
11. ReactJS [Електронний ресурс] - Режим доступу до ресурсу: <https://react.dev/reference/react>
12. Tailwind CSS [Електронний ресурс] - Режим доступу до ресурсу: <https://tailwindcss.com/docs>
13. YUP schema validation [Електронний ресурс] - Режим доступу до ресурсу: <https://github.com/jquense/yup>

14. React hook form_[Електронний ресурс] - Режим доступу до ресурсу:
<https://www.react-hook-form.com/>
15. Zustand [Електронний ресурс] - Режим доступу до ресурсу:
<https://docs.pmnd.rs/zustand/>
16. Tanstack query_[Електронний ресурс] - Режим доступу до ресурсу:
<https://tanstack.com/query/latest/>
17. ShadCN UI_[Електронний ресурс] - Режим доступу до ресурсу:
<https://ui.shadcn.com/docs>
18. Leaflet map_[Електронний ресурс] - Режим доступу до ресурсу:
<https://leafletjs.com/>
19. Lucide icons [Електронний ресурс] - Режим доступу до ресурсу:
<https://lucide.dev/icons/>
20. OpenAI [Електронний ресурс] - Режим доступу до ресурсу:
<https://platform.openai.com/docs/api-reference>
21. Нова пошта API - [Електронний ресурс] - Режим доступу до ресурсу:
<https://developers.novaposhta.ua/documentation>
22. Посвістак В. С. Клієнт-серверна архітектура та її використання при розробці програмного забезпечення / В. С. Посвістак, Т. І. Демківська // Інформаційні технології в науці, виробництві та підприємництві: збірник наукових праць молодих вчених, аспірантів, магістрів кафедри комп'ютерних наук та технологій / за заг. наук. ред. В. Ю. Щербаня. – Київ: Освіта України; ФОП Маслаков, 2020. – С. 78-81.
23. PostgreSQL- [Електронний ресурс] - Режим доступу до ресурсу:
<https://www.postgresql.org/>
24. GTmetrix - [Електронний ресурс] - Режим доступу до ресурсу:
<https://gtmetrix.com/>
25. NextJS Serever actions _-[Електронний ресурс] - Режим доступу до ресурсу:
<https://nextjs.org/docs/app/building-your-application/data-fetching/server-actions-and-mutations>

ДОДАТКИ

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						69

Лістинг файлу product.service.ts

```
@Injectable()
export class ProductService {
  strapiUrl: string;
  strapiKey: string;
  strapiEntityName = 'products';
  constructor(
    private readonly httpService: HttpService,
    private readonly configService: ConfigService,
  ) {
    this.strapiUrl = this.configService.get<string>('strapiUrl');
    this.strapiKey = this.configService.get<string>('strapiAccessKey');
  }

  sendRequest(requestParams: string) {
    return firstValueFrom(
      this.httpService
        .get(`${this.strapiUrl}/${this.strapiEntityName}${requestParams}`, {
          headers: {
            Authorization: `Bearer ${this.strapiKey}`,
          },
        })
        .pipe(
          catchError((e) => {
            console.error(e);
            throw new Error(e);
          }),
        ),
    );
  }

  async getProduct(productId: number) {
    const requestParams =
      `/${productId}?populate[logo]=*&populate[subcategory][populate][logo]=*`;

    const response = await this.sendRequest(requestParams);

    return parseProductStrapiData(response.data.data);
  }

  async getProducts({
    skip,
    take,
    search,
    fromPrice,
    toPrice,
    subCategoryId,
  })
```

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						70

```

    categoryId,
    order,
  }: GetProductsDto) {
    let requestParams =
`?populate[logo]=*&populate[subcategory][populate][logo]=*&pagination[start]={skip}
&pagination[limit]={take}`;
    if (subCategoryId) {
      requestParams += `&filters[subcategory][id]={subCategoryId}`;
    }
    if (categoryId) {
      requestParams += `&filters[subcategory][category][id]={categoryId}`;
    }
    if (fromPrice) {
      requestParams += `&filters[price][$gte]={fromPrice}`;
    }
    if (toPrice) {
      requestParams += `&filters[price][$lte]={toPrice}`;
    }
    if (search) {
      requestParams += `&filters[name][$containsi]={search}`;
    }
    if (order) {
      requestParams += `&sort=price:{order}`;
    }
    const response = await this.sendRequest(requestParams);
    return response.data.data.map((product) => parseProductStrapiData(product));
  }
}

```

Лістинг файлу main.ts

```

const app = await NestFactory.create(AppModule, {
  logger: ['log', 'error', 'warn', 'debug', 'verbose'],
  rawBody: true,
});

app.useGlobalPipes(new ValidationPipe({ transform: true }));
app.enableCors();
const logger = new Logger('Bootstrap');

const config = new DocumentBuilder()
  .setTitle('Codlo Research')
  .setDescription('Codlo research description')
  .setVersion('0.0.1')
  .addBearerAuth({
    description: `[just text field] Please enter token in following format: Bearer
<JWT>`,
    name: 'Authorization',
    bearerFormat: 'Bearer',
    scheme: 'Bearer',
    type: 'http',
    in: 'Header',
  });

```

```

    })
    .build();

    const document = SwaggerModule.createDocument(app, config);
    SwaggerModule.setup('api', app, document);

    const PORT = app.get(ConfigService).get('port');

    await app.listen(PORT, () => {
        logger.log(`Server started on ${PORT}`);
    });

```

Лістинг файлу layout.tsx

```

import Header from "@components/Header";
import Hydration from "@components/Hydration";
import Navigation from "@components/Navigation";
import NavigationMobile from "@components/NavigationMobile";
import UserMenu from "@components/UserMenu";
import { Toaster } from "@components/ui/toaster";
import { api } from "@lib/axios";
import { routes } from "@lib/routes";
import { cn } from "@lib/utils";
import type { Metadata } from "next";
import { Inter as FontSans } from "next/font/google";
import { cookies } from "next/headers";
import Image from "next/image";
import Link from "next/link";
import Cart from "../(main-pages)/_cart/Cart";
import "../globals.css";

const fontSans = FontSans({
  subsets: ["latin"],
  variable: "--font-sans",
});

export const metadata: Metadata = {
  title: "VetStore",
};

export async function updateCookie(token?: string) {
  "use server";
  cookies().set("vet-store-token", `${token}`, {
    httpOnly: true,
    path: "/",
  });
}

export default async function RootLayout({
  children,

```

```

}: Readonly<{
  children: React.ReactNode;
}>) {
  const token = cookies().get("vet-store-token");
  let isValidToken = false;
  try {
    await api.get("/user/me", {
      headers: { Authorization: `Bearer ${token?.value}` },
    });
    isValidToken = true;
  } catch (error) {
    isValidToken = false;
  }

  return (
    <html lang="en">
      <body className={cn("h-screen w-full font-sans ", fontSans.variable)}>
        <div className="flex h-full w-full flex-col items-center">
          <Hydration />

          <Header className="mx-6 flex h-full items-center justify-between">
            <Link
              href={routes.catalog}
              className="relative h-full w-16 md:w-20"
            >
              <Image src="/logo.png" fill alt="logo" unoptimized />
            </Link>

            <Navigation />
            <div className="flex items-center gap-4">
              <Cart />
              <UserMenu
                logOut={updateCookie}
                token={isValidToken ? token?.value : undefined}
              />
            </div>
          </Header>
          {children}
        </div>
        <Toaster />
      </body>
    </html>
  );
}

```

Лістинг файлу support.ts

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						73

```

const openai = new OpenAI({
  apiKey: process.env.CHAT_API_KEY,
});

export async function POST(req: NextRequest) {
  const body = (await req.json()) as { question: string };

  try {
    const completion = await openai.chat.completions.create({
      messages: [
        {
          role: "user",
          content: body.question,
        },
      ],
      model: "gpt-3.5-turbo",
    });

    return NextResponse.json(
      { answer: completion.choices[0].message.content },
      { status: 200 },
    );
  } catch {
    return NextResponse.json(
      { error: "Вибачте але чат зараз недоступний." },
      { status: 500 },
    );
  }
}

```

Лістинг файлу useBaseQuery.ts

```

import {
  QueryKey,
  UseQueryOptions,
  UseQueryResult,
  useQuery,
} from "@tanstack/react-query";
import { AxiosResponse } from "axios";

type IBaseQueryOptions<T> =
  | Omit<UseQueryOptions<T, unknown, T, QueryKey>, "queryKey" | "queryFn">
  | undefined;

function useBaseQuery<T>(
  queryKey: QueryKey,
  request: (signal: AbortSignal | undefined) => Promise<AxiosResponse<T, any>>,
  options?: IBaseQueryOptions<T>,
): UseQueryResult<T | undefined, unknown> {
  return useQuery({
    queryKey,

```

					ІПЗ.КР.Б-121-24-ПЗ	Арк.
						74

```

    queryFn: async function ({ signal }) {
      const response = await request(signal);
      return response.data;
    },
    ...options,
  });
}

```

```
export default useBaseQuery;
```

Лістинг файлу useProductQuery.ts

```

export const useProductsQuery = (
  params: Omit<IProductParams, "skip" | "take">,
  perPage: number,
) => {
  return useInfiniteQuery({
    queryKey: ["products", params, perPage],
    queryFn: async (p) => {
      const d = await getAllProducts({
        ...params,
        skip: p.pageParam * perPage,
        take: perPage,
      });

      return {
        data: d.data,
        next: p.pageParam + 1,
        prev: p.pageParam - 1,
        empty: d.data.length === 0,
        isLast: d.data.length < perPage,
      };
    },
    initialPageParam: 0,
    getNextPageParam: (lastPage) => lastPage.next,
    getPreviousPageParam: (firstPage) => firstPage.prev,
  });
};

```

Лістинг файлу cartStore.ts

```

import { create } from "zustand";
import { persist } from "zustand/middleware";
import {
  CartItem,
  addToCart,
  clearCart,
  deleteFromCart,
  getUserCart,
}

```

```

} from "../api/cartApi";

export interface ProductItem {
  id: number;
  quantity: number;
}

type Store = {
  isLogIn: boolean;
  items: CartItem[];
  updateCartItem: (p: CartItem) => void;
  deleteItem: (id: number) => void;
  deleteAll: () => void;
  isAdded: (id: number) => number | undefined;
  syncCarts: (state: boolean) => Promise<void>;
  setLogin: (state: boolean) => void;
  getTotal: () => number;
};

export const useCartStore = create<Store>()(
  persist(
    (set, get) => ({
      isLogIn: false,
      items: [],
      updateCartItem: async (newItem) => {
        set((state) => {
          const itemIndex = state.items.findIndex(
            (product) => product.id === newItem.id,
          );

          if (itemIndex === -1) {
            state.items.push(newItem);
          } else {
            state.items[itemIndex].quantity = newItem.quantity;
          }

          return { ...state };
        });
      },
      if (get().isLogIn)
        await addToCart({
          productId: newItem.id,
          quantity: newItem.quantity,
        });
    },
    deleteItem: async (id) => {
      set((state) => ({
        items: state.items.filter((item) => item.id !== id),
      }));
      if (get().isLogIn)
        await deleteFromCart({

```

```

        productId: id,
        quantity: 0,
    });
},
deleteAll: async () => {
    set(() => ({ items: [] }));

    if (get().isLogIn) {
        await clearCart();
    }
},
isAdded: (id) => {
    return get().items.find((item) => item.id === id)?.quantity ?? 0;
},
setLogin: (login) => {
    set((state) => ({ items: state.items, isLogIn: login }));
},
getTotal: () => {
    return +get()
        .items.reduce((acc, cur) => {
            return acc + cur.price * cur.quantity;
        }, 0)
        .toFixed(2);
},

syncCarts: async (login) => {
    set((state) => ({ items: state.items, isLogIn: login }));
    if (!login) {
        return;
    }

    const products = get().items;
    try {
        const { data } = await getUserCart();

        if (products.length > 0 && data.basketItems.length === 0) {
            const promises = products.map((item) =>
                addToCart({
                    productId: item.id,
                    quantity: item.quantity,
                })),
            );
            await Promise.all(promises);
        }
        if (data.basketItems.length > 0) {
            set((state) => ({
                ...state,
                items: [...data.basketItems],
            }));
        }
    }
}

```

```

        } catch {
            set((state) => ({ ...state }));
        }
    },
    }),
    {
        name: "vet-store-cart-storage",
        skipHydration: true,
    },
    ),
);

```

Лістинг файлу prApi.ts

```

import axios from "axios";

type CityType = "село" | "місто";
type WarehousesType = "Postomat" | "Store";

export interface CityItem {
    AreaDescription: string;
    CityID: string;

    Description: string;
    DescriptionRu: string;
    Ref: string;
    SettlementTypeDescription: CityType;
}

interface Cities {
    data: CityItem[];
}

export interface WarehousesItem {
    CategoryOfWarehouse: WarehousesType;
    CityDescription: string;
    CityRef: string;

    Description: string;
    Number: string;
    Ref: string;
    SettlementTypeDescription: CityType;
}

interface Warehouses {
    data: WarehousesItem[];
}

export const getCities = (city?: string) =>

```

```

axios.post<Cities>(process.env.NEXT_PUBLIC_NP_API_URL ?? "", {
  apiKey: process.env.NEXT_PUBLIC_NP_API_KEY,
  modelName: "AddressGeneral",
  calledMethod: "getCities",
  methodProperties: {
    Page: "1",
    FindByString: city,
    Limit: "80",
  },
});

export const getWarehouses = (city?: string, warehouses?: string) =>
axios.post<Warehouses>(process.env.NEXT_PUBLIC_NP_API_URL ?? "", {
  apiKey: process.env.NEXT_PUBLIC_NP_API_KEY,
  modelName: "AddressGeneral",
  calledMethod: "getWarehouses",
  methodProperties: {
    Page: "1",
    FindByString: warehouses,
    CityName: city,
    Limit: "80",
  },
});

```

Лістинг сторінки catalog

```

import PageContent from "./PageContent";

function Home() {
  return (
    <div className="animate-show flex h-full w-full flex-col items-center px-6 ">
      <h1 className="mt-24 text-6xl font-700">Каталог</h1>
      <PageContent />
    </div>
  );
}

export default Home;

```

Лістинг файлу PageContent.tsx

```

"use client";

import Loader from "@components/Loader";
import { Button } from "@components/ui/button";
import {
  Carousel,
  CarouselContent,
  CarouselItem,

```

```

    CarouselNext,
    CarouselPrevious,
  } from "@components/ui/carousel";
import { Input } from "@components/ui/input";
import { useCategoriesQuery } from "@lib/queries/categoriesQueries";
import { useProductsQuery } from "@lib/queries/productsQueries";
import { cn } from "@lib/utils";
import { Search } from "lucide-react";
import Image from "next/image";
import { Fragment, useDeferredValue, useState } from "react";
import CategorySkeleton from "../components/CategorySkeleton";
import EmptyProducts from "../components/EmptyProducts";
import FilterBar from "../components/FilterBar";
import MobileFilterBar from "../components/MobileFilterBar";
import ProductCard from "../components/ProductCard";

export type SortingOption = { value: "asc" | "desc"; label: string };

export const sortingOptions: SortingOption[] = [
  { value: "asc", label: "Ціна за зростанням" },
  { value: "desc", label: "Ціна за спаданням" },
];

function PageContent() {
  const [selectedCategory, setSelectedCategory] = useState<number>();
  const [selectedSubCategories, setSelectedSubCategories] = useState<string[]>(
    [],
  );

  const [selectedSorting, setSelectedSorting] = useState<SortingOption>();

  const [minMaxPrices, setMinMaxPrices] = useState<number[]>([0, 9999]);

  const [searchText, setSearchText] = useState<string>("");
  const deferredSearch = useDeferredValue(searchText);

  const categoriesQuery = useCategoriesQuery();

  const productsQuery = useProductsQuery(
    {
      fromPrice: minMaxPrices[0],
      toPrice: minMaxPrices[1],
      categoryId: selectedCategory,
      subCategoryId: selectedSubCategories[0]
        ? +selectedSubCategories[0]
        : undefined,
      search: deferredSearch,
      order: selectedSorting?.value,
    },
    12,
  );

```

```

);
const pages = productsQuery.data?.pages ?? [];

const categoriesData = categoriesQuery.data ?? [];

return (
  <>
    <div className=" mb:my-24 my-12 flex w-full justify-center gap-4">
      <Carousel
        opts={{
          align: "start",
        }}
        className=" w-full md:max-w-2/3 "
      >
        <CarouselContent className="">
          {categoriesQuery.isLoading && <CategorySkeleton />}

          {categoriesData?.map((item) => (
            <CarouselItem
              key={item.id}
              className={cn(
                "basis-1/2 justify-center md:basis-1/3 lg:basis-1/4 xl:basis-1/5",
                categoriesData.length < 5 &&
                `xl:basis-1/${categoriesData.length === 1 ? "2" :
categoriesData.length}`,
              )}
            >
              <Button
                onClick={() =>
                  setSelectedCategory(
                    selectedCategory === item.id ? undefined : item.id,
                  )
                }
                variant="clear"
                className={cn(
                  "relative mx-auto flex h-36 w-36 flex-col items-center justify-
center gap-3 rounded-full bg-gray-100 p-6 hover:bg-green-primary hover:text-white",
                  selectedCategory === item.id &&
                  "bg-green-primary text-white",
                )}
              >
                <Image
                  src={item.logo}
                  width={100}
                  height={100}
                  alt="category-logo"
                  className=" h-1/2 w-1/2 select-none text-white"
                  loading="lazy"
                />
                <p className=" text-base font-600 ">{item.name}</p>
              </div>
            </CarouselItem>
          )}
        </CarouselContent>
      </div>
    </>
  )

```

```

        </Button>
      </CarouselItem>
    )})
  </CarouselContent>
  <CarouselPrevious className=" hidden md:inline-flex " />
  <CarouselNext className=" hidden md:inline-flex " />
</Carousel>
</div>
<div className="mb-12 flex w-full flex-col justify-between gap-4 md:flex-row
md:justify-end ">
  <MobileFilterBar>
    <FilterBar
      categories={categoriesData}
      selectedSubCategories={selectedSubCategories}
      setSelectedSubCategories={setSelectedSubCategories}
      values={minMaxPrices}
      max={9999}
      min={0}
      onChange={setMinMaxPrices}
      selectedSorting={selectedSorting}
      setSelectedSorting={setSelectedSorting}
    />
  </MobileFilterBar>
  <div className="w-full rounded-8 bg-white px-4 py-2 shadow-sm md:w-1/3
md:max-w-80 md:self-end">
    <Input
      className=" pl-0 "
      containerClassName="w-full"
      labelClass="pointer-events-none"
      type="text"
      label="Шукати товар..."
      adorenment={<Search className=" pointer-events-none my-auto " />}
      value={searchText}
      onChange={(e) => setSearchText(e.target.value)}
    />
  </div>
</div>

<div className="flex w-full gap-6">
  <div className="sticky top-6 hidden h-fit w-1/4 flex-col gap-4 rounded-16
bg-white px-8 py-4 shadow-sm md:flex">
    <FilterBar
      categories={categoriesData}
      selectedSubCategories={selectedSubCategories}
      setSelectedSubCategories={setSelectedSubCategories}
      values={minMaxPrices}
      max={9999}
      min={0}
      onChange={setMinMaxPrices}
      selectedSorting={selectedSorting}
    >

```

```

        setSelectedSorting={setSelectedSorting}
      />
    </div>
    <div className=" flex h-fit w-full flex-wrap items-center justify-evenly
gap-4 md:w-3/4  ">
      {pages.map((page, index) => (
        <Fragment key={index}>
          {page.data?.map((item) => (
            <ProductCard key={item.id} product={item} />
          ))}
        </Fragment>
      ))}
      {!productsQuery.isLoading &&
        !productsQuery.isFetching &&
        pages[0]?.empty && <EmptyProducts />}

      {(productsQuery.isLoading || productsQuery.isFetching) && <Loader />}
      <div className="flex w-full justify-end p-8">
        <Button
          variant="clear"
          className=" flex items-center gap-2 self-start rounded-full bg-green-
primary px-4 py-2 text-base font-600 text-white hover:bg-green-primary/80"
          onClick={() => productsQuery.fetchNextPage()}
          disabled={pages[pages?.length - 1]?.isLast}
        >
          Завантажити ще...
        </Button>
      </div>
    </div>
  </div>
</>
);
}

export default PageContent;

```