МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____Юрій КОНДРАТЕНКО

«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБЗАСТОСУНОК ДЛЯ ПРОДАЖУ ПРОДУКТІВ ХАРЧУВАННЯ

Спеціальність 122 «Комп'ютерні наук»

Освітня програма «Комп'ютерні науки»

Здобувач _____ Олена ТРОФИМЕНКО

«___»_____ 2025 р.

Керівник ст. викл. кафедри КІ _____Євген ДАРНАПУК

«___»_____червня_____2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри інтелектуальних інформаційних систем
_____Юрій КОНДРАТЕНКО
«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Трофименко Олена Євгенівна
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Вебзастосунок для продажу продуктів харчування».

Керівник роботи: Дарнапук Євген Сергійович, ст. викл. кафедри КІ

Затверджена наказом ЧНУ ім. Петра Могили від «18» листопад 2024 р. №321

2. Строк представлення кваліфікаційної роботи «11» _червня_ 2025 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: розробка повноцінного вебзастосунку інтернет-магазину з реалізацією основного функціоналу, що забезпечує зручну взаємодію між користувачем і системою.

4. Перелік питань, що підлягають розробці: дослідження предметної області та аналіз існуючих аналогів; формування специфікації вимог до ПЗ; визначення архітектури для проектування програмного забезпечення; моделювання та проектування програмного забезпечення; розробка програмного забезпечення;

здійснення тестування роботи програмного забезпечення; проведення аналізу результатів розробки.

5. Перелік графічних матеріалів: презентація.

Керівник роботи _____ Євген ДАРНАПУК
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Олена ТРОФИМЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «18» грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Вебзастосунок для продажу продуктів харчування

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	18.12.2024	28.12.2024	Виконано
2	Аналіз предметної області та постановка задачі	15.01.2024	01.02.2024	Виконано
3	Огляд літератури та аналогічних систем щодо вебзастосунку для продажу харчових продуктів.	03.02.2025	20.02.2025	Виконано
4	Огляд існуючих застосунків для вирішення поставленої задачі	19.04.2025	22.04.2025	Виконано
5	Реалізація обраних технологій з аналізом отриманих результатів	22.04.2025	03.05.2025	Виконано
6	Перший попередній захист КР на засіданні комісії кафедри	16.05.2025	16.05.2025	Виконано
7	Корегування роботи за результатами попереднього захисту	17.05.2025	20.05.2025	Виконано
8	Другий попередній захист КР на засіданні комісії кафедри	30.05.2025	30.05.2025	Виконано
9	Доробка та остаточне оформлення КР	01.06.2025	03.06.2025	Виконано
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	11.06.2025	12.06.2025	Виконано

Керівник роботи _____ Євген_ДАРНАПУК
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Олена ТРОФИМЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«30» січня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувачки групи 402 ЧНУ ім. Петра Могили

Трофименко Олени Євгенівни

на тему: **«ВЕБЗАСТОСУНОК ДЛЯ ПРОДАЖУ ПРОДУКТІВ ХАРЧУВАННЯ»**

Актуальність роботи полягає у розробці вебзастосунку для продажу продуктів харчування, що є актуальним напрямом у контексті розвитку електронної комерції. Сучасні цифрові технології дозволяють забезпечити користувачам швидкий та зручний доступ до широкого асортименту продуктів без необхідності фізичного відвідування торгових закладів. Використання вебзастосунку сприяє оптимізації процесу пошуку, вибору та оформлення замовлень, що підвищує ефективність комерційної діяльності та рівень задоволеності споживачів.

З огляду на стрімке зростання популярності онлайн-покупок, розробка вебзастосунку набуває особливої актуальності. Вебзастосунок передбачає інтеграцію з базами даних, використання алгоритмів фільтрації, а також забезпечення адаптивного інтерфейсу для різних пристроїв. Такі рішення підвищують доступність продуктів харчування для покупців із усього світу, зокрема людей з обмеженою мобільністю та жителів віддалених регіонів.

Об’єкт роботи є процес розробки вебзсастосунку для продажу продуктів харчування.

Предмет роботи є інформаційні технології та методи створення вебзастосунків для електронної комерції.

Метою роботи є розробка універсального вебзастосунку для продуктів харчування з фільтрацією товарів і підключенням до бази даних, який адаптований до зручного перегляду, пошуку та замовлення продуктів харчування онлайн.

Бакалаврська кваліфікаційна робота складається з вступу, 3 розділів, висновків, переліку джерел посилань та додатку.

У вступі визначається актуальність теми, об’єкт, предмет та мета роботи, завдання які необхідно виконати. У першому розділі проаналізовано існуючи застосунки-аналоги, визначено їх функціональні можливості, переваги та недоліки. Наведено аналіз подібних рішень і сформульовано специфікацію вимог.

У другому розділі описується огляд мов, технологій та бібліотек, що використовуються для розробки, проєктування програмного забезпечення, аналізу ключових можливостей вебзастосунку, процесу обробки даних, а також формулюванню специфікації проєкту.

У третьому розділі описано процес розробки програмного забезпечення, продемонстровано функціонал вебзастосунку та проведено тестування.

Кваліфікаційна робота бакалавра викладена на 90 сторінок, містить 3 розділи, 52 ілюстрацій, 17 таблиць, 30 джерел в переліку посилань, додаток А.

Ключові слова: *створення вебзастосунку, респонсивний інтерфейс, основна технологія Node.js, database, опис продукції, відгуки, каталог, фільтрація, пошук, помилка, зворотній зв’язок, кошик.*

ABSTRACT

to the qualification work by the student of the group 402 of Petro Mohyla Black Sea National University

Trofymenko Olena

«WEB APPLICATION FOR FOOD SALES»

The relevance of the work consists of the development of a web application for the sale of food products, which is a relevant direction in the context of the development of e-commerce. Modern digital technologies allow users to provide quick and convenient access to a wide range of products without the need to physically visit retail outlets. Using a web application helps optimize the process of searching, selecting and placing orders, which increases the efficiency of commercial activities and the level of consumer satisfaction.

Therefore, given the rapid growth in the popularity of online shopping, the development of such a web application becomes particularly relevant. The web application involves integration with databases, the use of filtering algorithms, as well as providing an adaptive interface for different devices. These solutions improve the accessibility of food products to buyers worldwide, including people with limited mobility and residents of remote areas.

The object of the work is the process of developing a web application for selling food products.

The subject of the work is information technologies and methods for creating web applications for e-commerce.

The purpose of the work is to develop a universal web application for food products with product filtering and connection to a database, which is adapted for convenient viewing, searching and ordering food products online.

The bachelor's qualification work consists of an introduction, 3 sections, conclusions, a list of references and an appendix.

The introduction defines the relevance of the topic, the object, subject and purpose of the work, the tasks that need to be performed. The first section analyzes existing similar applications, determines their functionality, advantages and disadvantages. An analysis of similar solutions is presented and a specification of requirements is formulated.

The second section describes an overview of languages, technologies and libraries used for development, software design, analysis of key features of a web application, data processing process, as well as formulation of a project specification.

The third section describes the software development process, demonstrates the functionality of the web application and conducts testing.

The third section describes the process of software development and testing.

The bachelor's thesis is 90 pages long, contains 3 chapters, 52 illustrations, 17 table, 30 sources in the list of references, and appendix A.

Keywords: *web application creation, responsive interface, core Node.js technology, database, product description, reviews, catalog, filtering, search, error, feedback, cart.*

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

СУБД	–	Система керування базами даних
БД	–	Бази даних
ПЗ	–	Програмне Забезпечення
COVID-19	–	Коронавірусна хвороба 2019 року
HTML	–	HyperText Markup Language
HTML5	–	Hypertext Markup Language 5
CSS	–	Cascading Style Sheets
SEO	–	Search Engine Optimization
FTP	–	File Transfer Protocol
API	–	Application programming interface

ВСТУП

Актуальність теми. Технологічний прогрес в наш час не стоїть на місці, і тепер можна, не виходячи з дому через телефон, планшет, ноутбук або комп'ютер, лише в один клік у вебзастосунку, купити все необхідне для комфортного життя. Це сприяє зменшенню скупчення великої кількості людей як у магазинах, так і на паркувальних місцях біля них, але при цьому збільшує кількість робочих місць у супермаркетах, адже тепер потрібні працівники, які можуть зібрати товар і передати тому, хто його забере. Через це покупки стали зручними, як для покупців усередині будівлі, так і для тих, хто вирішив придбати щось онлайн.

Особливу актуальність ця тема набула в умовах пандемії COVID-19 і зараз, оскільки люди прагнуть заощадити час, який пов’язаний із поїздками до фізичних магазинів. Крім того, користування вебзастосунком спрощує процес вибору товарів завдяки зручному інтерфейсу та своєчасному оновленню асортименту, що сприяє більш раціональному плануванню покупок.

Об’єкт роботи є процес розробки вебзсастосунку для продажу продуктів харчування.

Предмет роботи є інформаційні технології та методи створення вебзастосунків для електронної комерції.

Метою кваліфікаційної роботи є розробка універсального вебзастосунку для продуктів харчування з фільтрацією товарів і підключенням до бази даних, який адаптований до зручного перегляду, пошуку та замовлення продуктів харчування онлайн.

Для досягнення поставленої мети необхідно виконати наступні завдання:

- проаналізувати предметну сферу роботи;
- розглянути існуючи аналоги;
- розробити макет;
- створити дизайн користувацького інтерфейсу;

- використати основну технологію;
- додати зв’язок з базою даних;
- протестувати роботу вебзастосунку;
- реалізувати логіку фільтрації та сортування товарів;
- забезпечити зручну навігацію між категоріями та підкатегоріями;
- адаптувати вебзастосунок для різних пристроїв;
- перевірити коректність виводу даних з бази;
- виправити помилки та оптимізувати;
- сформувати інструкцію для користувача.

1 АНАЛІЗ СУЧАСНИХ ВЕБЗАСТОСУНКІВ ІНТЕРНЕТ-МАГАЗИНІВ

1.1 Особливості онлайн-продажу продуктів харчування

Інтернет-магазин – це вебзастосунок, додаток або платформа, яка дозволяє компаніям продавати свої товари чи послуги онлайн. Клієнти можуть переглядати товари, робити покупки та здійснювати платежі онлайн, що зручніше, доступніше та має ширший асортимент товарів, ніж традиційні роздрібні торговці [1]. Вони стали важливою частиною сучасного ринку, дозволяючи компаніям отримувати доступ до клієнтів по всьому світу. Пошукова оптимізація (SEO), ефективне управління продуктами та маркетинг – все це важливі компоненти прибуткового інтернет-магазину. Фізичні магазини без онлайн-присутності більше не достатньо. Компанії з найкращою онлайн-присутністю, особливо ті, що дозволяють клієнтам купувати безпосередньо через інтернет, залучають клієнтів у потрібний час. Ці компанії, мають свій вебзастосунок, які оптимізовані для пошукових систем, розуміють, як інвестувати в цифровий маркетинг, і визнають, що чим кращий користувацький досвід, тим більша ймовірність того, що клієнти оберуть саме їхні товари, а не товари конкурентів. На відміну від традиційних компаній, інтернет-магазин не має «робочого часу», оскільки він завжди відкритий. Але у інтернет-магазинів є свої переваги та недоліки. А саме до переваг можна віднести:

– без тиску покупців або продавців;
– різноманітність пропозицій;
– зручність;
– не витрачай час.

До недоліків можна віднести:

– проблеми з доставкою;
– неможливість перевірки товару;
– невідповідний термін придатності.

Веброзробка включає всі етапи створення вебпроєкту або вебзастосунку, від стратегії та дизайну до тестування та розгортання. Для виконання різних завдань, пов'язаних з розробкою вебзастосунку, потрібно зробити декілька етапів.

1. Початковий етап веброзробки – це створення стратегії. Він включає постановку цілей та завдань, створення списку справ.

2. Дизайн та специфікація – після встановлення стратегії наступним кроком у веброзробці є створення плану. Розробник повинен встановити терміни та параметри. Завдання цього етапу такі: створення підходу передбачає планування основного контенту, створення ескізного дизайну, створення макета на основі надійного дизайну, створення прототипу проєкту та тестування прототипу.

3. Веброзробник повинен враховувати всі аспекти дизайну та функціонування сайту під час створення вебпроєкту. Це включає розробку фронтенду та бекенду.

4. Тестування та обслуговування – це життєво важливий етап веброзробки, який гарантує виконання всіх поставлених вимог. Потім перевіряється код та тестуються нові функції. Це може бути зроблено вручну або автоматично, залежно від типу тестування та вебпроєкту. Якщо результат поганий, вживається заходи для усунення шкідників.

5. Реєстрація у інтернет-провайдера – вебпроєкт має бути зареєстрований у інтернет-провайдера. Після чого проєкт завантажується на сервер. Вебзастосунок розміщується через FTP (протокол передачі файлів), створений обліковий запис вебпроєкту призначається певний доменний простір на сервері інтернет-провайдера.

6. Останній крок – запуск: це завершальний етап процесу розробки. Проєкт запускається після реєстрації у інтернет-провайдера та його публічного доступу. Основні обов'язки на цьому етапі: міграція даних, запуск сервера, об'єднання коду та перенаправлення доменних імен.

Кожна з цих фаз має важливе значення для успішного завершення проєкту і забезпечення досягнення поставлених цілей. Визначені вимоги дозволяють

перейти до встановлення архітектури системи та її впровадження відповідно до встановлених наших поставлених цілей та задач.

1.2 Аналіз застосунків-аналогів

На сьогоднішній день є чи мало аналогів інтернет-магазину харчового товару та не тільки. Для розгляду візьмемо декілька прикладів вітчизняних, іноземних та інтернаціональних аналогів. До вітчизняних можна віднести:

- АТБ-Маркет;
- Сільпо;
- ЕкоМаркет;
- Таврія В.

До інтернаціональних аналогів можна віднести:

- METRO;
- Ашан.

До іноземних аналогів можна віднести:

- Walmart;
- Coscto;
- Amazon Fresh;
- Sam`s Club;
- Smart & Final;
- Target.

Проаналізуємо більш детально кожний. Спочатку про вітчизняні.

1. АТБ-Маркет (див. рис. 1.1). Це найбільша мережа дисконтних магазинів в Україні, що пропонує великий вибір товарів за доступними цінами. Компанія постійно розширює свою мережу, яка тепер включає мобільний додаток та вебсайт, де клієнти можуть переглядати пропозиції та товари [2].

2. Сільпо (див. рис. 1.2). Це один з провідних мереж продуктових магазинів України, відомий своїм широким асортиментом, відмінним сервісом та креативним

дизайном магазинів. Компанія також пропонує розумний мобільний додаток та інтернет-платформу для замовлення товарів з доставкою [3].

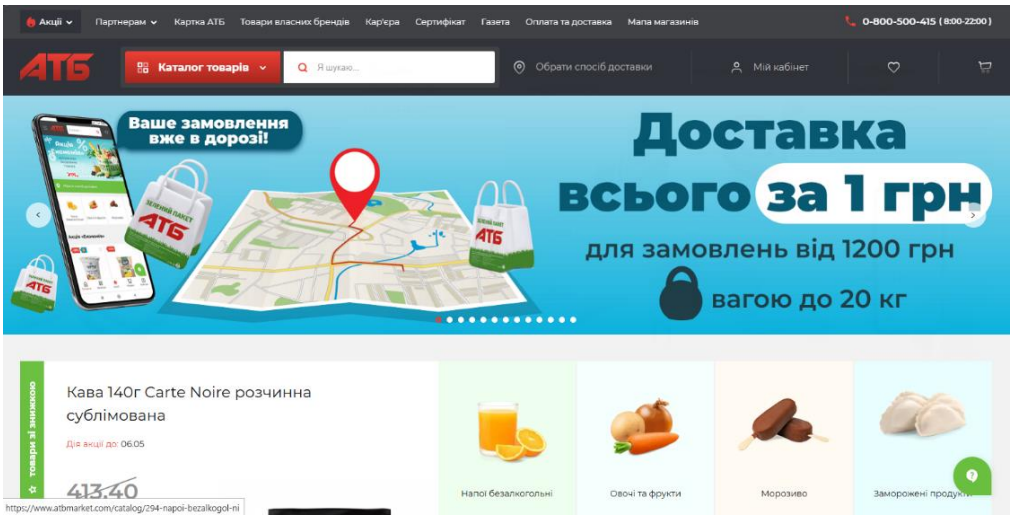


Рисунок 1.1 – АТБ-Маркет

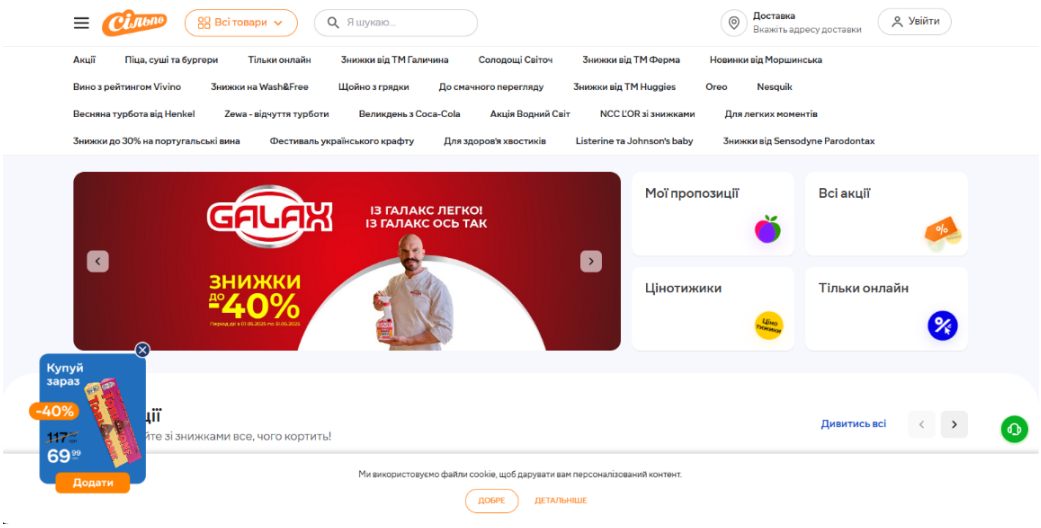


Рисунок 1.2 – Сільпо

3. ЕкоМаркет (див. рис. 1.3). Українська мережа продуктових магазинів, заснована у 2003 році, має понад 80 магазинів у 15 регіонах України. Фірма продає різноманітний вибір продуктів харчування, побутової хімії та інших товарів, які дуже необхідні за вигідною вартістю, а також магазин має власну кухню та пекарню [4].

4. Таврія В (див. рис. 1.4). Це загальнонаціональна продуктова компанія, заснована в 1992 році, має понад 100 магазинів у восьми регіонах України. Фірма продає різноманітний асортимент товарів, включаючи власні бренди, та пропонує зручні онлайн-покупки через інтернет-магазин та мобільний додаток з доставкою до ваших дверей [5].

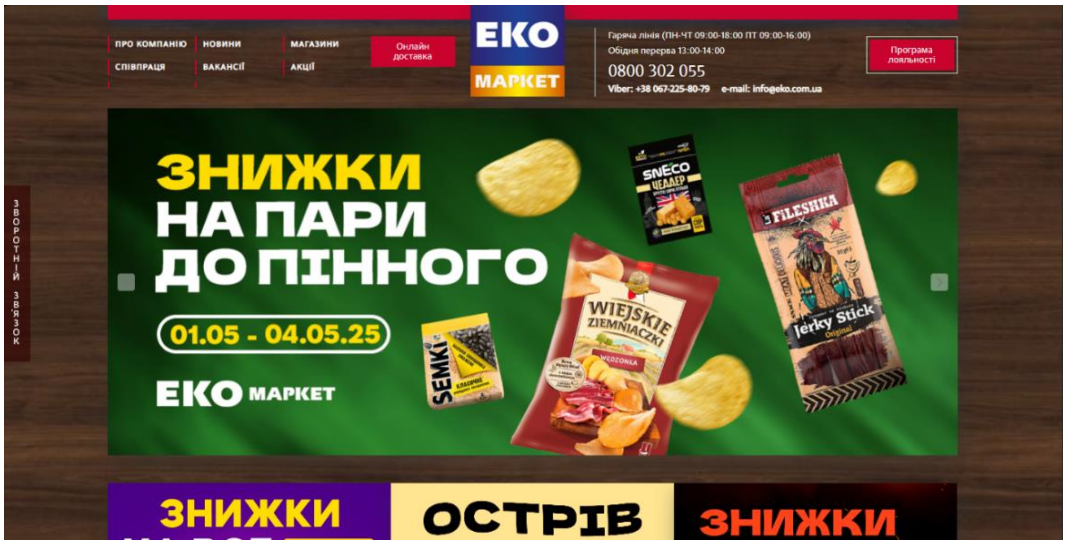


Рисунок 1.3 – ЕкоМаркет

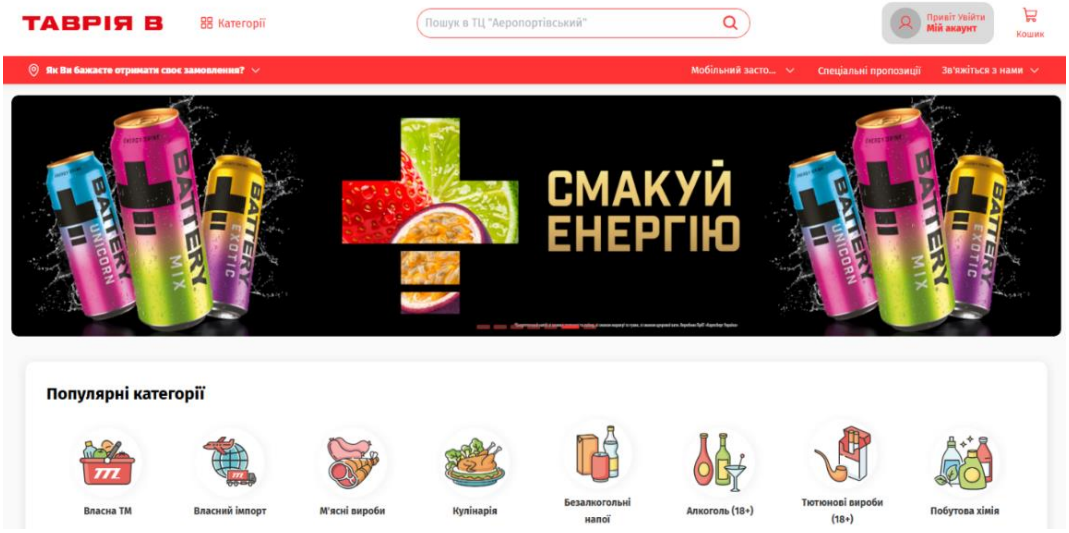


Рисунок 1.4 – Таврія В

5. Метро (див. рис. 1.5). Це всесвітня оптова мережа, яка працює в Україні у форматі готівкового розрахунку, обслуговуючи як бізнес, так і роздрібних споживачів. Компанія пропонує платформу онлайн-замовлень, яка дозволяє клієнтам вибирати між доставкою та самовивозом [6].

6. Ашан (див. рис. 1.6). Це багатонаціональна мережа гіпермаркетів, яка продає різноманіття товарів за доступною вартістю, таких як продукти харчування, побутова хімія та меблі. Фірма має інтернет-магазин, де клієнти можуть розміщувати замовлення з доставкою або самовивозом у місцевому магазині [7].

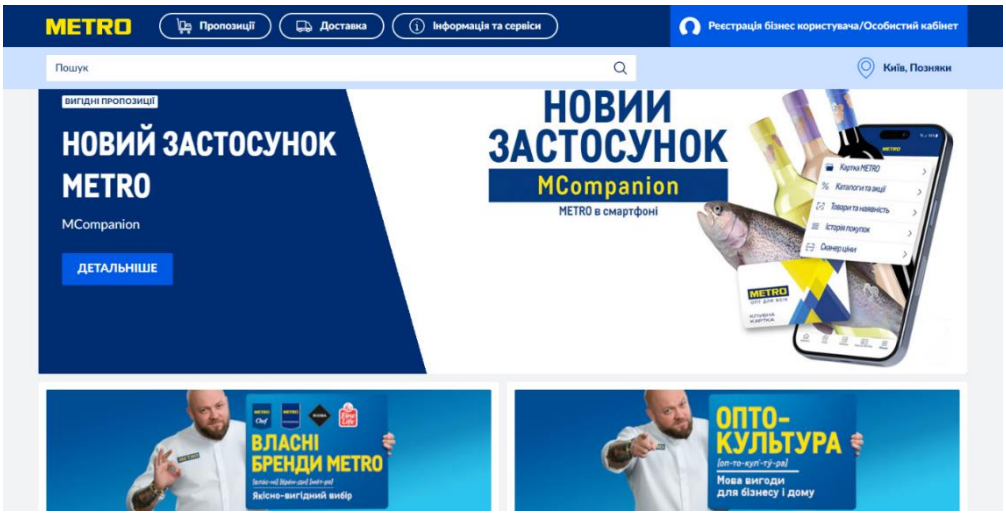


Рисунок 1.5 – Метро

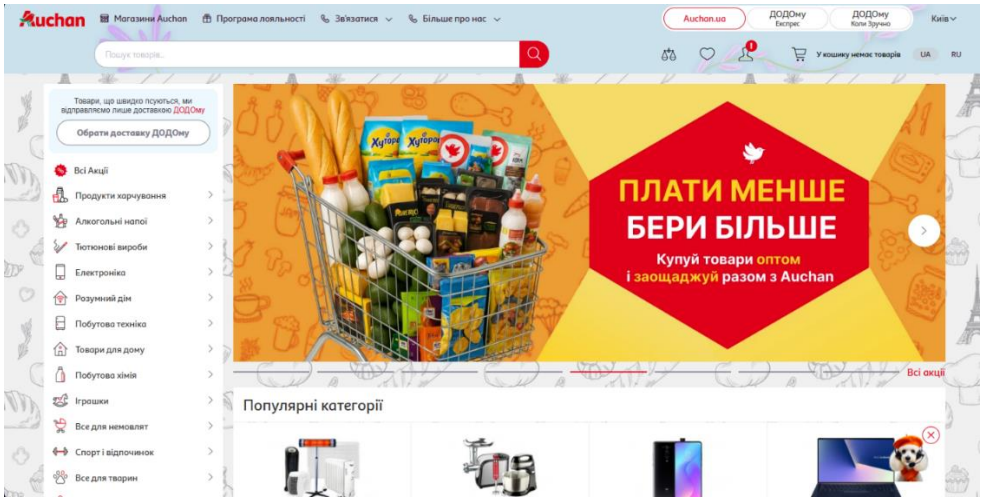


Рисунок 1.6 – Ашан

7. Walmart (див. рис. 1.7). Це найбільша у світі роздрібна компанія з понад 10 000 магазинів у 19 країнах та близько 270 мільйонами клієнтів щотижня. Компанія швидко розвиває онлайн-сервіси, такі як мобільний додаток, доставка продуктів, програма лояльності Walmart+ та передові можливості, такі як сканування товарів у магазині за допомогою смартфона [8].

8. Coscto (див. рис. 1.8). Це міжнародна мережа складських магазинів, яка працює на основі членства, пропонуючи товари у великих кількостях за зниженими цінами. Компанія відома власним брендом Kirkland Signature та надає онлайн-послуги, такі як доставка продуктів і мобільний додаток для зручних покупок [9].

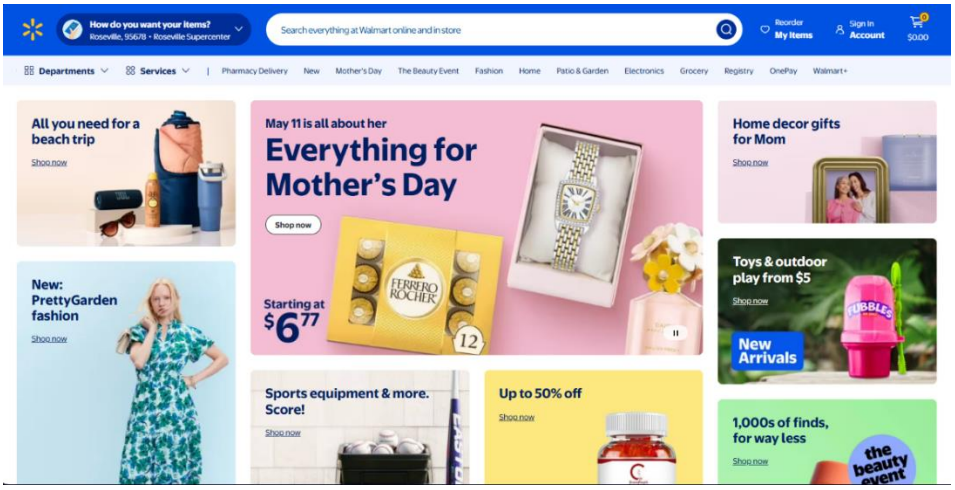


Рисунок 1.7 – Walmart

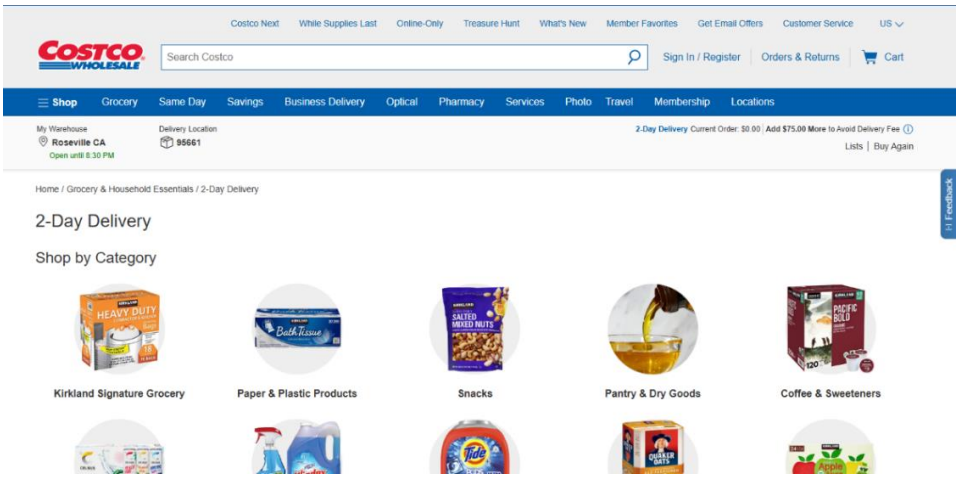


Рисунок 1.8 – Coscto

9. Amazon Fresh (див. рис. 1.9). Це сервіс доставки продуктів лише для учасників Amazon Prime, який пропонує доставку різноманітних свіжих та упакованих продуктів у той самий день. Amazon Fresh також має фізичні магазини в деяких містах США та Європи, автоматизуючи процес покупок за допомогою сучасних технологій, таких як Dash Cart [10].

10. Sam`s Club (див. рис. 1.10). Американська компанія-склад, що працює на основі членства, заснована в 1983 році та належить Walmart. Мережа пропонує своїм понад 69 мільйонам учасників різноманітний вибір товарів зі знижкою, а також онлайн-послуги, такі як доставка, самовивіз та додаток для смартфонів для швидких покупок [11].

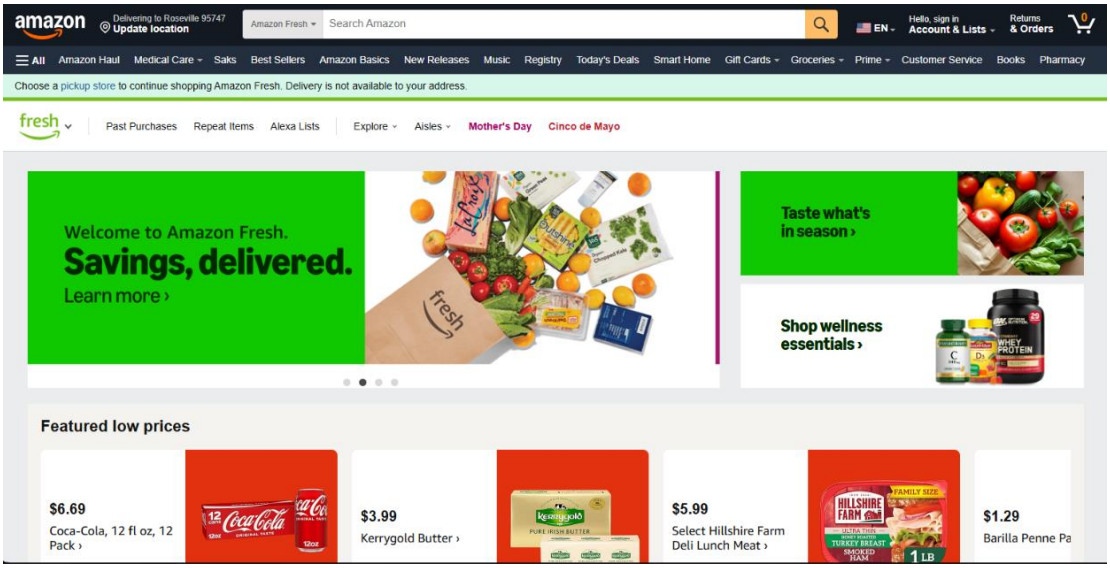


Рисунок 1.9 – Amazon Fresh

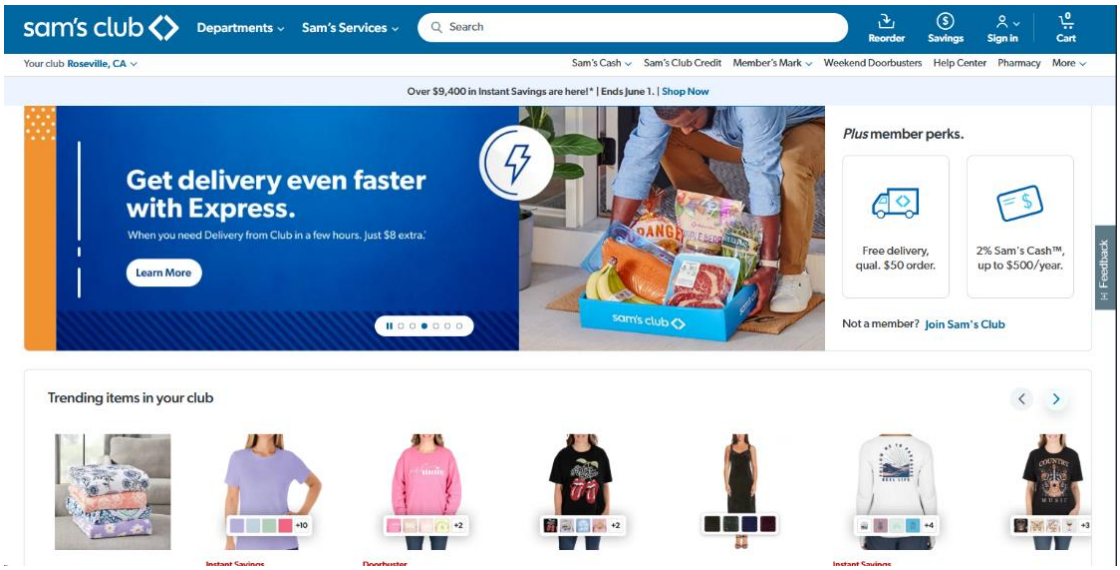


Рисунок 1.10 – Sam`s Club

11. Smart & Final (див. рис. 1.11). Це американська мережа складських магазинів, заснована в 1871 році, яка пропонує широкий асортимент товарів, включаючи свіжі фрукти, м'ясо та понад 3000 товарів у великих упаковках, без необхідності членства. Бізнес обслуговує як приватних, так і комерційних клієнтів з понад 250 магазинів у Каліфорнії, Неваді, Аризоні та північній Мексиці, а також здійснює покупки онлайн для зручності [12].

12. Target (див. рис. 1.12). Це один із найбільших мереж у Сполучених Штатах, який має майже 2000 магазинів та різноманітний асортимент продукції, що включає продукти харчування, електроніку, речі та предмети домашнього вжитку. Компанія добре відома своїми доступними цінами та вишуканою елегантністю, і активно розширює свої онлайн-пропозиції, включаючи мобільний додаток, доставку та самовивіз [13].

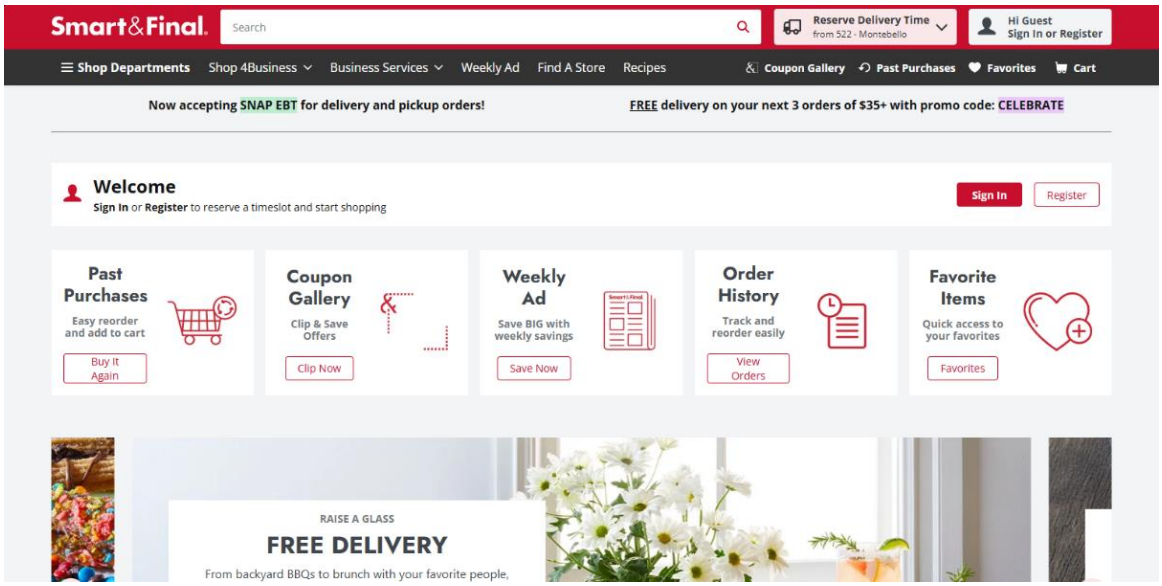


Рисунок 1.11 – Smart & Final

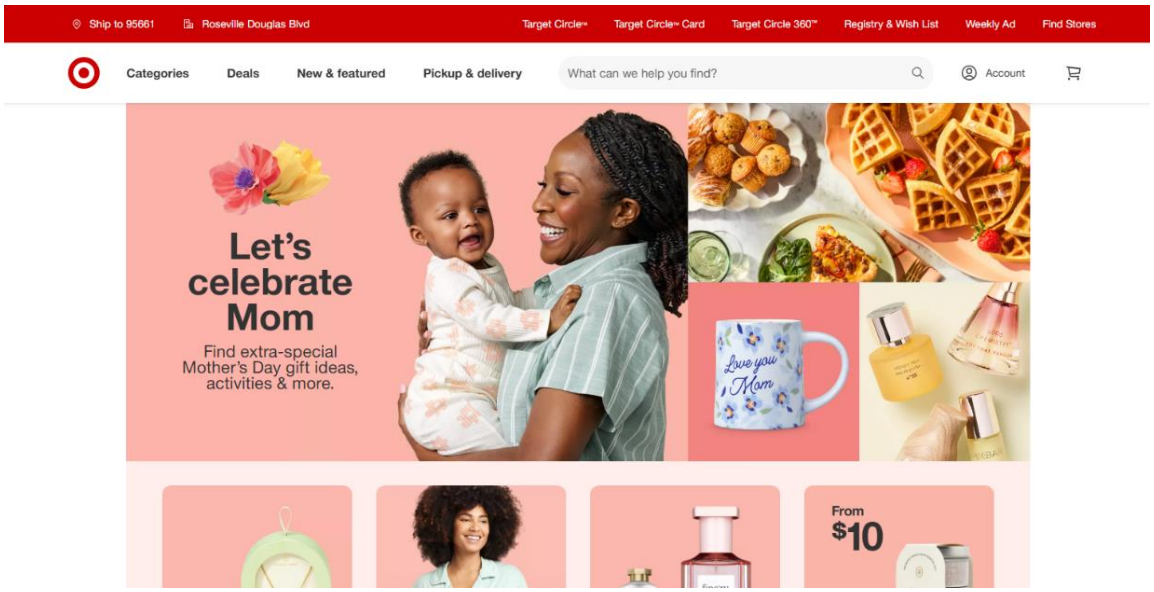


Рисунок 1.12 – Target

Такі мережі, як АТБ-Маркет, Сільпо, ЕкоМаркет і Таврія В, є ключовими онлайн-магазинами в бізнесі, маючи віддану клієнтську базу та значний вплив на українських споживачів. Для аналізу було використано порівняльну таблицю (див. в табл. 1.1).

Таблиця 1.1 – Порівняльний аналіз показників діяльності торговельних мереж

Критерії	**АТБ-Маркет**	**Сільпо**	**ЕкоМаркет**	**Таврія В**
Ціновий рівень	+	-	+	+
Неактуальна інформація	-	-	-	+
Можливість зворотного зв’язку	-	+	-	+
Повільна обробка замовлень	-	+	-	+
Застарілий дизайн	+	-	+	-
Можливість збільшувати фото	-	+	-	+
Зручність використання	-	+	-	+

Отже, в результаті порівняльного аналізу можна виділити, що найліпшим із усіх магазинів є Сільпо. А найгіршими є АТБ-Маркет, ЕкоМаркет.

Walmart, Costco, Amazon Fresh, Sam's Club, Smart & Final та Target є одними з найвідоміших онлайн-магазинів, кожен з яких має свої переваги та недоліки. Для аналізу було використано порівняльну таблицю (див. в табл. 1.2).

Таблиця 1.2 – Аналіз інформаційної подачі в онлайн-магазинах

Критерії	**Walmart**	**Costco**	**Amazon Fresh**	**Sam's Club**	**Smart & Final**	**Target**
Ціновий рівень	+	+	+	+	+	+
Неактуальна інформація	-	+	-	-	-	+

Кінець таблиці 1.2

Критерії	**Walmart**	**Costco**	**Amazon Fresh**	**Sam's Club**	**Smart & Final**	**Target**
Можливість зворотного зв’язку	-	+	-	-	-	+
Повільна обробка замовлень	-	+	-	-	-	+
Застарілий дизайн	+	-	+	-	-	-
Можливість збільшувати фото	-	+	-	+	+	+
Членська картка	-	+	-	+	-	-
Зручність використання	-	+	-	+	+	+

В результаті порівняльного аналізу можна виділити, Costco пропонує недорогі ціни на великі кількості товарів та високоякісний власний бренд, але покупки доступні лише членам клубу, так само як і в Sam's Club. Walmart, Amazon Fresh, приваблює клієнтів різноманітним вибором товарів, доступними цінами та швидкою системою доставки, але періодично має перевантаження сервісу, що знижує швидкість обслуговування. Smart & Final та Target має хороший баланс між ціною, якістю та зручністю покупок, і активно розробляє мобільний додаток та сервіс доставки, проте його вибір продуктів харчування поступається, ніж у спеціалізованих компаній.

1.3 Специфікація вимог інтернет-магазину продуктів харчування

Призначенням вебзастосунку є зручний онлайн доступ до асортименту товарів. Це дає користувачеві швидко обирати та замовляти потрібні продукти харчування. Система спрямована зробити процес придбання товарів максимально

простим, швидким і комфортним для клієнта. Досягається автоматизація вебзастосунку та покращується взаємодія між покупцем та сервісом. У процесі роботи було узгоджено, що для ПЗ використовувалась основна технологія – Node.js.

Сфера застосування:

Вебзастосунок призначений для всіх хто не хоче нікуди їхати на закупки, або хто не має на це часу, також для тих хто хоче побачити реальні скидки та весь асортимент продуктів з оцінкою від покупців. Для цього потрібен лише телефон, планшет, ноутбук або комп'ютер, головне щоб він був підключений до Інтернету.

Структурна організація програмного рішення:

1. Клієнтська частина (Front-end):

– HTML5 – структура;

– CSS – стиль і макет вебсторінки;

– JavaScript – динамічна клієнтська логіка.

2. Серверна частина (Back-end):

– здійснює обробку запитів, підтримує логіку програми та контролює доступ до даних. Створено за допомогою Node.js, серверного середовища виконання JavaScript.

3. База даних:

– MongoDB – це документоорієнтована NoSQL база даних для зберігання та аналізу даних.

Ключові можливості вебзастосунку, обробка інформації та специфікація проєкту.

1. Вебзастосунок розроблений для забезпечення повного функціоналу інтернет-магазину, такого як реєстрація користувачів, управління замовленнями, аналітика.

2. Система дозволяє реєстрацію та авторизацію користувачів, що вимагає надання особистої інформації. Авторизація здійснюється електронною поштою або іншим унікальним ідентифікатором.

3. Для полегшення навігації включено спосіб пошуку та фільтрації товарів за назвою, категорією, брендом, ціною, рейтингом, наявністю акцій та іншими критеріями. На основі активності користувачів інтегрована система рекомендацій рекомендує схожі або пов'язані товари.

4. Товари в каталозі організовані за категоріями та підкатегоріями. Кожен товар має детальний опис, технічні характеристики, фотографії та відгуки.

5. Кошик дозволяє додавати товари, змінювати їх кількість або видаляти. Вартість товарів розраховується автоматично з урахуванням податків, знижок та вартості доставки. Покупець може зберегти кошик та оформити замовлення пізніше.

6. Управління запасами здійснюється за допомогою автоматизованої системи, яка перевіряє наявність товарів, змінює кількість у відповідь на запити та інформує склад про необхідність поповнення запасів.

7. Додаток захищає персональні дані, запобігає несанкціонованому доступу та шифрує конфіденційну інформацію.

8. Щоб забезпечити внесок клієнтів, було додано такі методи зв'язку, як електронна пошта та телефонна підтримка.

Отже, ця специфікація вимог надає огляд функцій та можливостей програмного забезпечення для інтернет-магазинів продуктів харчування. Під час проєктування та розробки ці критерії можуть бути оновлені та розширені. Створення інтернет-магазину продуктів вимагає комплексного плану, який гарантує належну реалізацію всіх функцій та вимог, зазначених у специфікації.

Висновки до першого розділу

У першому розділі роботи було описано предметну сферу: інтернет-магазин – це вебзастосунок, додаток або платформа, яка дозволяє компаніям продавати свої товари чи послуги, опис його переваг та недоліків, застосування. Також оглянуто застосунки-аналоги. Такі мережі, як АТБ-Маркет, Сільпо, ЕкоМаркет і Таврія В, є ключовими онлайн-магазинами в бізнесі, маючи віддану клієнтську базу та

значний вплив на українських споживачів. В результаті порівняльного аналізу можна виділити, що найліпшим із усіх магазинів є Сільпо. А найгіршими є АТБ-Маркет, ЕкоМаркет. Потім було оглянуто: Walmart, Costco, Amazon Fresh, Sam's Club, Smart & Final та Target, кожен з яких має свої переваги та недоліки. В результаті порівняльного аналізу можна виділити: Costco пропонує недорогі ціни на великі кількості товарів та високоякісний власний бренд, але покупки доступні лише членам клубу, так само як і в Sam's Club. Walmart, Amazon Fresh, приваблює клієнтів різноманітним вибором товарів, доступними цінами та швидкою системою доставки, але періодично має перевантаження сервісу, що знижує швидкість обслуговування. Smart & Final та Target має хороший баланс між ціною, якістю та зручністю покупок, і активно розробляє мобільний додаток та сервіс доставки, проте його вибір продуктів харчування поступається, ніж у спеціалізованих компаній.

Зазначено властивості та функції проєктної частини: клієнти можуть переглядати товари, робити покупки та залишати відгуки, що зручніше та доступніше і має ширший асортимент товарів, ніж традиційні магазини. Використання онлайн-магазину є зручний онлайн доступ до асортименту товарів. Це дає користувачеві швидко обирати та замовляти потрібні продукти харчування.

2 МОДЕЛІ ТА МЕТОДИ ДЛЯ ВИРІШЕННЯ ЗАДАЧ КЛАСИФІКАЦІЙ

2.1 Вибір технологій для вирішення задач класифікацій

Для створення сайту використовувалась мова гіпертекстової розмітки HTML5, форма мова опису зовнішнього вигляду вебсторінки CSS та мова програмування JavaScript, кожна з яких має свої переваги та недоліки. Ось декілька з них:

– HTML5;
– CSS;
– JavaScript;
– Express-фреймворк;
– Node.js.

HTML, тобто HyperText Markup Language – це стандартна мова розмітки, яка призначена для використання з ранніми вебдокументами. HTML-документи можна конвертувати в інші типи вебсторінок [14]. Коли документ створюється за допомогою перевіреного HTML, веббраузер може використовувати HTML для відображення основних елементів і підзаголовків документа. Більшість документів мають стандартні елементи, такі як заголовки, абзаци або списки. Вебсайти на основі HTML можна використовувати, щоб зрозуміти певні елементи, надати вебдодаткам математично обґрунтовану інформацію для створення представлень елементів, вставити інформацію в документи тощо. Усе необхідне для читання HTML-документа у веббраузері, який декодує HTML-код і відображає на екрані користувача документ, який ідентифікує автора. Для аналізу переваг та недоліків використано (див. в табл. 2.1).

HTML5 – це мова розмітки для структури та подання вмісту Всесвітньої мережі [15]. HTML5 підтримує традиційний синтаксис у стилі HTML і XHTML та інші нові функції в його розмітці. HTML5 використовується веббраузерами для візуалізації коду. Він містить кілька покращень у можливостях вебсайтів, розробці вебконтенту тощо. Для аналізу переваг та недоліків використано (див. в табл. 2.2).

Таблиця 2.1 – Порівняльна таблиця переваг та недолік HTML

Категорія	**Перевага**	**Недолік**
Простота вивчення	Простий синтаксис, багато навчальних ресурсів для початківців	Не має
Кросплатформеність	Працює на всіх сучасних браузерах і ОС.	Різна підтримка нових стандартів у старих браузерах
Структура та семантика	Семантичні теги покращують доступність і структуру контенту.	Не має
Інтеграція з іншими технологіями	Легка інтеграція з CSS та JavaScript	Залежність від CSS і JS для стилів і функціональності
Інтерактивність і динаміка	Не має	Обмежена взаємодія без JavaScript; відсутність динамічного оновлення
Оформлення	Не має	Відсутність внутрішніх стилів, потреба в зовнішньому CSS для дизайну

HTML є невід'ємною частиною веброзробки та забезпечує структурування та фундамент для основи вебсторінок.

Таблиця 2.2 – Порівняльна таблиця переваг та недолік HTML5

Перевага	**Недолік**
Нові семантичні теги для покращення структури та SEO	Часткова підтримка в старих браузерах
Працює на всіх сучасних браузерах і ОС	Різна підтримка нових стандартів у старих браузерах
Вбудовані теги для аудіо та відео без потреби в плагінах	Ризики безпеки через легкий доступ до localStorage
Розширені API (геолокація, офлайн-зберігання, drag & drop)	Складність реалізації сучасних API для початківців
Покращена кросплатформеність і адаптація під мобільні	Залежність від інших технологій (CSS3, JavaScript, API)
Підтримка офлайн-режиму через localStorage і Service Workers	Деякі старі технології, як applicationCache, застаріли

CSS – це абревіатура мови каскадних таблиць стилів, яка призначена для оформлення елементів, створених за допомогою мов розмітки, таких як HTML [16].

Він ізолює вміст сайту від його візуального зображення. HTML і CSS нерозривно пов'язані, оскільки HTML є основою сайту, а CSS – це образність всієї сторінки. Для аналізу переваг та недоліків використано (див. в табл. 2.3).

Таблиця 2.3 – Порівняльна таблиця переваг та недолік CSS

Перевага	**Недолік**
Розділення змісту та представлення – дозволяє змінювати стиль без змін у HTML-структурі	Складність вивчення – новачкам складно розібратись у синтаксисі та механіці роботи з HTML
Повторне використання стилів – класи й ідентифікатори можна застосовувати багаторазово	Кросбраузерна сумісність – різні браузери по-різному обробляють CSS
Гнучкість у стилізації – можна налаштувати кольори, розміри, типи шрифтів, відступи, структуру макетів та інше	Обмежені функції інтерактивності – складні анімації чи логіку реалізувати лише на CSS складно
Підтримка адаптивного дизайну – легко розробляти застосунки, які мають гарну адаптацію на будь-яких пристроях	Залежність від структури HTML – для ефективного використання CSS потрібно мати добре структурований код

Отже, CSS є потужним інструментом оформлення та візуального представлення вебсторінок. Незважаючи на певні обмеження, він дозволяє створювати сучасний і привабливий дизайн, підтримувати єдність стилю та адаптивність інтерфейсу, що значно покращує взаємодію користувачів із сайтом.

Для реалізації системи було обрано мову програмування JavaScript. Вона поєднує властивості об'єктно-орієнтованого підходу з унікальною реалізацією через прототипне наслідування, що відрізняє її від класичних ООП-мов. Окрім цього, JavaScript підтримує функціональні концепції, такі як функції першого класу, анонімні функції, замикання та каррінг, що забезпечує їй високу гнучкість і багатофункціональність.

Javascript – це динамічна мова програмування, яка легко освоюється і широко застосовується для створення інтерактивних вебсторінок. Вона дозволяє сценаріям на стороні клієнта взаємодіяти з користувачем, формуючи динамічний контент [17]. Мова є інтерпретованою та підтримує об'єктно-орієнтовані концепції. Основне призначення – розробка мережевих додатків. Вона доповнює та інте-

грується з Java, а також тісно взаємодіє з HTML. Мова є відкритою та кросплатформенною.

Попри наявність певних обмежень, JavaScript залишається потужним і широко застосовуваним засобом для розробки вебзастосунків. Його ключові переваги – висока масштабованість, можливість створення інтерактивного та динамічного контенту, а також велика спільнота розробників, яка підтримує і розвиває його функціонал. Для аналізу переваг та недоліків використано (див. в табл. 2.4).

Таблиця 2.4 – Порівняльна таблиця переваг та недолік Javascript

Перевага	**Недолік**
Висока розширюваність – дозволяє створювати як прості скрипти, так і складні вебдодатки	Залежність роботи від клієнтського середовища – різна підтримка в браузерах
Працює на клієнті й сервері – підтримка як у браузерах, так і на сервері (Node.js)	Проблеми безпеки – можливі XSS-атаки та інші вразливості клієнтського коду
Інтерактивність і динаміка – створення форм, анімацій, асинхронного завантаження даних	Обмежений доступ до системи файлів – неможливість напряму працювати з файлами на пристрої
Велике співтовариство – багато бібліотек, фреймворків і документації	Продуктивність – нижча у порівнянні з деякими мовами при обробці великих обсягів даних

Express-фреймворки є ефективними інструментами для розробки [18] вебзастосу-нків, які пропонують розробникам готові рішення та організовану структуру для продуктивної роботи з JavaScript. Для аналізу переваг та недоліків використано (див. в табл. 2.5).

Node.js – дуже потужний фреймворк/платформа на основі JavaScript, реалізовано на основі движку Google Chrome JavaScript V8 [19]. Він використовується для розробки вебдодатків із інтенсивним вводом-виводом, таких як сайти потокового відео, одно сторінкові додатки та інші вебдодатки. Node.js відкрито вихідний код, повністю безкоштовний і використовується тисячами розробників у всьому світі.

Таблиця 2.5 – Порівняльна таблиця переваг та недолік Express-фреймворки

Перевага	**Недолік**
Швидкість розробки	Мінімалізм функцій
Гнучка структура	Відсутність структури
Велика спільнота	Висока відповідальність (на розробнику)
Простота використання	Ручне конфігурування

Нижче наведено певні функції, які визначають Node.js пріоритетним вибором для архітекторів програмного забезпечення (див. рис. 2.1).

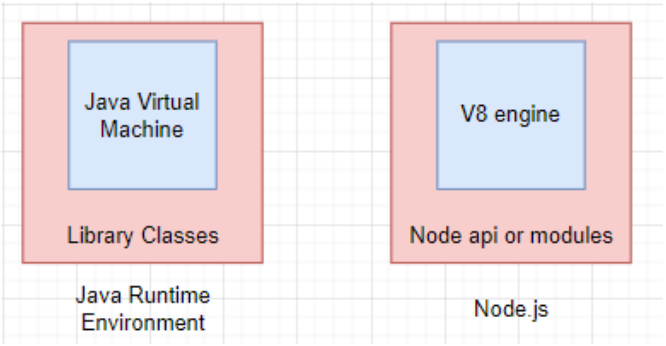


Рисунок 2.1 – Ілюстрація схеми Node.js

1. Асинхронний і подієво орієнтований – всі API бібліотеки Node.js працюють у асинхронному режимі.

2. Неблокуючий, мається на увазі, що сервер на базі Node.js ніколи припиняє роботу в очікуванні результатів.

3. API з поверненням результатів. Сервер продовжує роботу після виклику API, а подієвий механізм Node.js забезпечує отримання відповіді від попереднього виклику.

4. Висока продуктивність. Бібліотека Node.js, побудована на рушії Google Chrome V8 JavaScript Engine, забезпечує швидке виконання коду.

5. Однопотоковий, але високоефективний – Node.js застосовує однопотокову модель із циклом подій. Завдяки подієвому механізму сервер обробляє запити неблокуючим способом і забезпечує високу масштабованість, на відміну від традиційних серверів, які створюють обмежену кількість потоків для обробки запитів. Однопотокова архітектура Node.js дозволяє обслуговувати значно більше

запитів, ніж класичні сервери, наприклад HTTP-сервер Apache. Немає буферизації – програми Node.js ніколи не буферизують дані. Ці програми просто виводять дані пакетами.

6. Ліцензія − Node.js надана на умовах ліцензії MIT.

2.2 Методи та структура вебзастосунку

Метод – це функція, пов’язана з об’єктом і викликається через передачу повідомлень [20]. Об'єкт складається з даних про стан та поведінку, які разом забезпечують інтерфейс, що описує, як об'єкт може бути використаний. Методи дають змогу ізолювати логіку виконання та забезпечують засоби взаємодії з об’єктами в межах програми.

У процесі розробки вебзастосунку використовувались методи:

- аутентифікації користувачів;
- каталогізації товарів;
- обробки замовлень;
- захисту персональних даних;
- валідації даних;
- кешування клієнтських даних.

Структура – це спосіб групування кількох пов’язаних змінних разом в одній області [21]. Змінні структури називаються членами. Структура, на відміну від масиву, може містити різноманітні типи даних (див. рис. 2.2).

У проєкті наведено основну структуру вебзастосунку:

– перехід до “Домашня сторінка”: користувач може повернутися до головної сторінки сайту з будь-якого розділу;

– перегляд пропозиції у розділі “Магазин”: користувач потрапляє на сторінку з товарами, де доступний до великий вибір каталогу продуктів;

- “Про нас”: розділ містить інформацію про компанію, її історію, місію, команду працівників та основні пропоновані товари;
- “Серіс”: тут описані наявні послуги, які пропонує компанія;

– “Команда”: профілі ключових членів команди з короткими біографіями та контактною інформацією;

– “Контакти”: якщо у користувача є питання, проблеми або потрібна допомога – служба підтримки завжди на зв’язку;

– “Проєкт”: звідки йде постачання та хто власник продуктових ферм для магазину;

– “Новини”: актуальні новини та оновлення, пов’язані з продуктами харчування, подіями компанії.

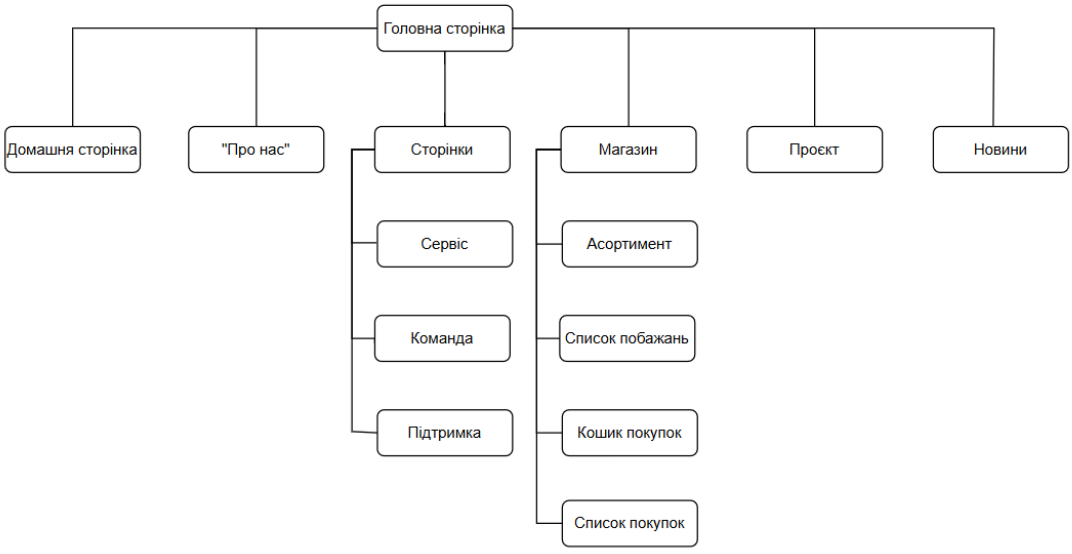


Рисунок 2.2 – Структура вебзастосунку

Отже, структура демонструє взаємозв’язки між основними модулями проєкту.

2.3 Інтеграційні процеси та сервіси користувацької підтримки

Для створення серверної частини вебзастосунку застосовано платформу Node.js, фреймворк Express.js та мову програмування JavaScript. Сервер розроблено за допомогою Node.js та Express, наявні роутер та контролер, що надають кінцеві точки API для використання у клієнтській частині застосунку. До серверу можна

виконувати POST та GET запити за кінцевими точками, а передача та повернення інформації відбувається за допомогою зручного для використання у JavaScript формату json.

Під час реєстрації дані користувача проходять перевірку за допомогою бібліотеки express-validator. У разі недотримання визначених правил форматування система повертає повідомлення про помилку й не передає запит далі до контролера. Якщо перевірка пройдена, на основі моделі створюється новий користувач з унікальними полями електронної пошти та номера телефону. Під час входу система порівнює введені користувачем облікові дані – пошту та пароль – із записами у базі даних. У разі успішної автентифікації сервер генерує токен доступу та передає його клієнтській частині. Усі наступні запити від цього користувача міститимуть токен, що підтверджує його автентичність. Якщо в базі вже є обліковий запис із вказаною поштою або номером телефону, система повідомляє про це користувача.

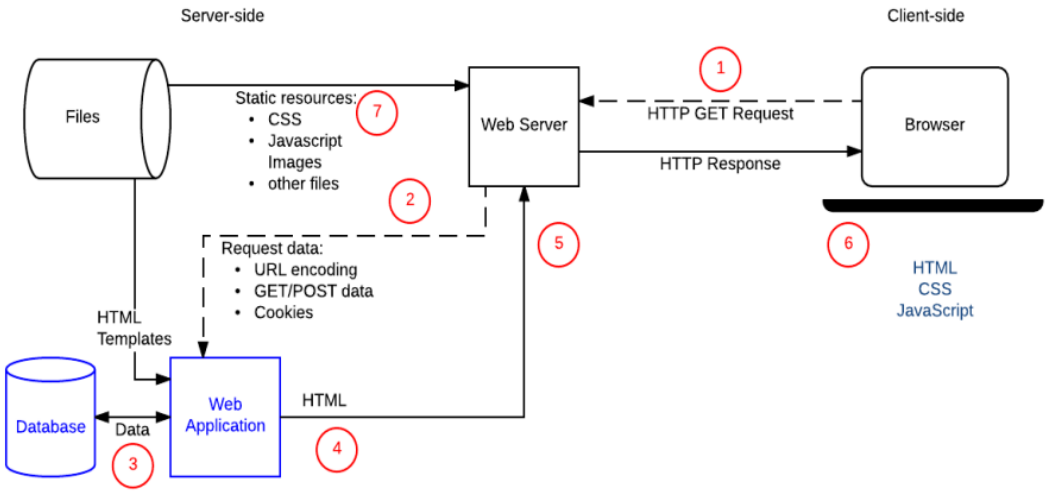


Рисунок 2.3 – Схема клієнт-серверної архітектури [22]

Вебзастосунок реалізований за клієнт-серверною архітектурою, де роль клієнта виконує веббраузер (див. рис. 2.3). Структура застосунку складається з трьох основних рівнів: інтерфейс користувача (браузер), серверна частина та БД.

2.4 Алгоритм пошуку та фільтрація товарів

У системі реалізовано функціонал пошуку товарів за ключовими словами, а також фільтрацію за кількома критеріями: категорія, підкатегорія, ціна, рейтинг, тощо. Ці фільтри формуються динамічно залежно від обраної категорії товарів.

Для розширення можливостей персоналізації пошуку передбачено впровадження алгоритмів рекомендаційних систем. Залежно від типу вхідних даних та взаємодії користувача з платформою, можуть застосовуватись два основних підходи:

– фільтрація за змістом: базується на властивостях товарів, які раніше зацікавили користувача (наприклад, категорія, ціновий діапазон). Система формує профіль користувача й підбирає подібні товари;

– колаборативна фільтрація: аналізує дії багатьох користувачів і виявляє схожість у поведінці. Наприклад, якщо кілька користувачів цікавилися однаковими товарами, система може рекомендувати їх іншим з подібною активністю. Недоліком колаборативної фільтрації є проблема: складність надання точних рекомендацій новим користувачам або новим товарам через нестачу даних.

Підготовча частина складається з наступних кроків.

1. З Dataset отримати теги товарів та провести їхню обробку: токенізацію та лематизацію (приведення різних граматичних форм слова до однієї форми).

2. Зберегти створений список тегів для кожного товару в окремий файл для подальшого перевикористання.

3. У майбутніх версіях вебзастосунку планується створення векторів частоти входження слів (за допомогою TF або TF-IDF).

4. Також передбачається застосування методу k-найближчих сусідів (k-NN) для обчислення схожості між товарами на основі векторів тегів.

5. Виконати ранжування товарів за ступенем подібності до цільового товару.

6. Зберегти отримані вектори та модель для подальшого використання.

Після виконання підготовки метод швидко працює в реальному часі.

1. Dataset товарів фільтрується так, щоб в ньому не було товару за ціною нижче ніж 20 із 50.

2. Завантажується раніше збережений список тегів, який поєднується з тегами цільового товару для формування вхідних даних.

3. У подальшому планується реалізувати обробку текстових тегів товарів із застосуванням токенізації та лематизації.

4. Для побудови векторних уявлень тегів буде використано методи TF-IDF, що дозволить застосовувати алгоритм k-найближчих сусідів (k-NN) для оцінки схожості товарів.

2.5 Реалізація категорій та підкатегорій

Категорія – це група товарів, які споживачі сприймають як пов’язані між собою або взаємозамінні. Асортимент розподіляють на такі категорії, враховуючи вподобання та потреби цільової аудиторії.

Підкатегорія – це об’єкт певної групи в межах категорії.

Товари зберігаються в “Product”, де є назва продукт, назва категорії та підкатегорії, до якої належить продукт. Кожен товар пов’язаний з певною підкатегорією, яка належить до основної категорії. Наприклад, товар банан належить до підкатегорії фрукти, яка входить до категорії продукту харчування. Таким чином, кожен товар прив’язується до конкретної категорії або підкатегорії, що спрощує адміністрування каталогу та реалізацію багаторівневої фільтрації на інтерфейсі користувача.

Отже, можна створити велику кількість категорії та підкатегорій, що забезпечить зручнішу та більш інформативну навігацію як для покупців, так і для адміністратора вебзастосунку.

2.6 Алгоритм роботи автоматизованої системи

Автоматизована підсистема обліку запасів здійснює динамічний моніторинг товарних залишків, реагує на транзакції в системі та автоматично генерує повідомлення при досягненні критичних рівнів на складі (див. рис. 2.4).

Основна логіка автоматизованої системи побудована на поетапній обробці змін запасів та контролі критичних рівнів залишків:

– система отримує запит на зміну продуктів;
– оновлення БД;
– контроль залишку продуктів;
– формується сповіщення до складу;
– очікування рішення.

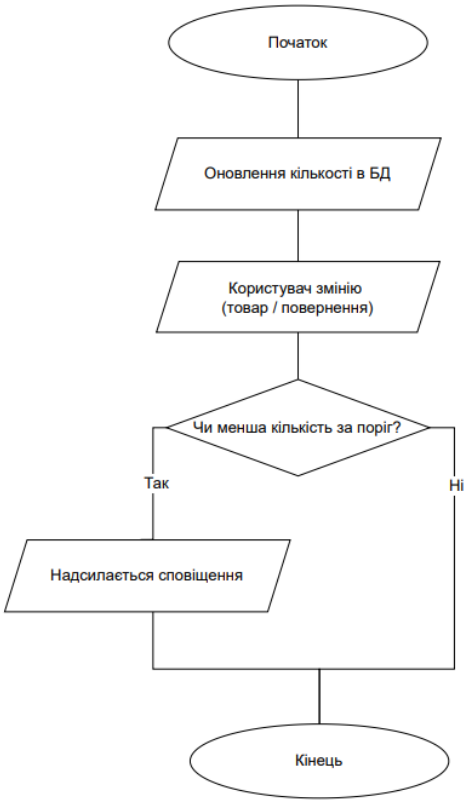


Рисунок 2.4 – Схема алгоритму автоматизованої системи

Завдяки автоматизації процесів контролю запасів знижується ризик виникнення дефіциту, скорочується час реакції на зміни, а також підвищується ефективність роботи складу та менеджменту.

2.7 Шифрування та захист персональних даних

Визначення користувача у поточній сесії виконується за допомогою jwt токена. При авторизації користувача до jwt токена додається ім’я, електронна адреса та ідентифікаційний код користувача, який буде використовуватись для його визначення в поточній сесії під час запиту до серверу з переданням токену в заголовках. Токен зберігається в локальному сховищі при авторизації. При виході користувача з платформи за допомогою кнопки “log out” значення токена звідти видаляється.

До цього всього в роботі відбувається шифрування за алгоритмом:

Система забезпечує надійний контроль безпеки персональних даних. Паролі користувачів не зберігаються у відкритому вигляді, а захищаються за допомогою відповідних методів шифрування за алгоритмом SHA-256 з додаванням salt, що унеможливлює відновлення навіть у разі витоку. Усі дані між клієнтом і сервером передаються через захищене з’єднання HTTPS. Це виключає будь-яке перехоплення чи зміну інформації. На рівні доступу діє ролева система: тільки авторизовані користувачі мають права на перегляд або зміну конкретних даних. Додатково система захищена від загроз, небезпечного введення і логує всі важливі дії.

Також, щоб користувачі міг легко зв’язатися, до сайту додано зручну форму для звернень. Через сторінку "Contact" кожен може залишити відгук, написати скаргу або просто поставити запитання. Після заповненням форми користувачем, його повідомлення автоматично надсилається на електронну пошту служби підтримки. У листі одразу видно, хто написав, про що йдеться, і як із цією людиною можна зв’язатися. Коли звернення надходить, його реєструють і передають відповідальному працівнику. Далі ми намагаємося якнайшвидше відповісти, щоб

допомогти або вирішити проблему. Така система дозволяє нам підтримувати хороший зв’язок із користувачами й дбати про якість сервісу.

2.8 Структура БД

MongoDB - це система управління БД, яка працює з документальною моделлю даних [23]. На відміну від реляційних СУБД, MongoDB не вимагає таблиць, схем або окремої мови запитів (див. рис. 2.5). Інформація зберігається у форматі документів або колекцій.

Рисунок 2.5 – Модель БД

Під час розробки для роботи з моделями та взаємодії з базою даних MongoDB використовувалися основні методи Mongoose, такі як:

- ProductsService - робота зі сховищем products;
- Mongoose.Schema() - функція, що задає схему моделі та описує її структуру;
- FeedbacksService - робота зі сховищем feedbacks;
- BlogArticlesService - робота зі сховищем blogarticles;
- MessagesService - робота зі сховищем messages;
- SubscribersService - робота зі сховищем subscribers;

- FarmsService - робота зі сховищем farms;
- TeammatesService - робота зі сховищем teammates;
- CartsService - робота зі сховищем carts;
- OrdersService - робота зі сховищем orders.

На діаграмі (див. рис. 2.6) показані класи, що описують моделі БД, та їх залежність від класу Schema пакета "mongoose". Mongoose – пакет Node.js, що спрощує взаємодію з хмарною БД MongoDB, а клас Schema надає інтерфейс для пошуку, збереження, оновлення та видалення документів у БД. Класи Feedback, Farm, BlogArticle, Message, Order, Cart, Subscriber, Product та Teammate описують сутність БД є прикладом ООП в БД. Всі вони успадковуються від класу Mongoose.Schema, що дозволяє взаємодіяти з БД через них, спрощуючи та прискорюючи роботу.

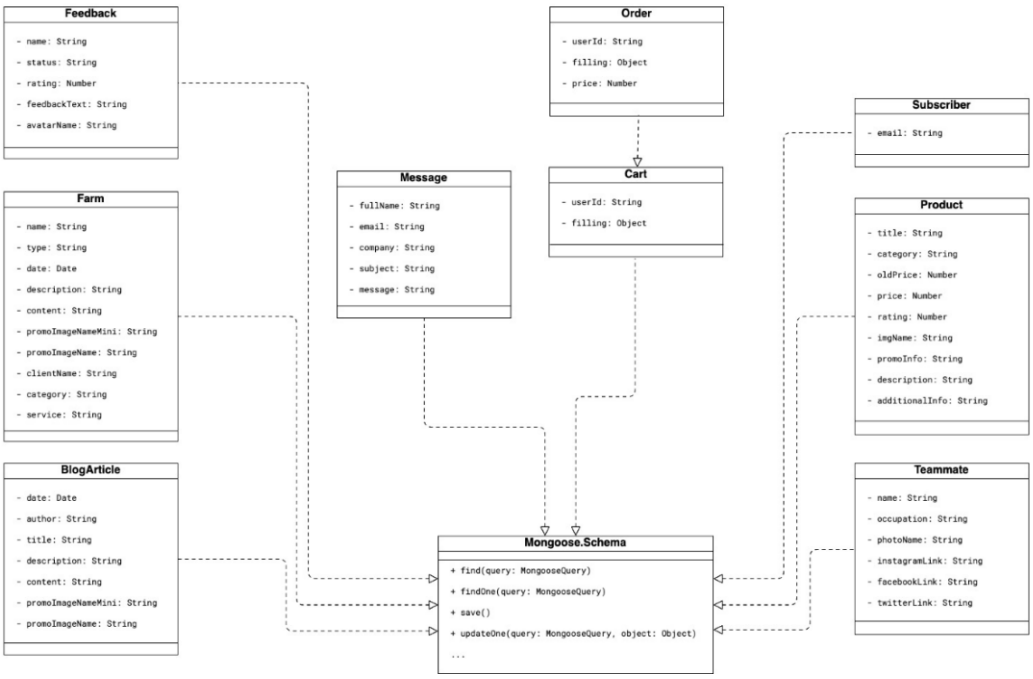


Рисунок 2.6 – Діаграма класів послуг бази даних

На діаграмі (див. рис. 2.7) показані класи, які забезпечують взаємодію програми-сервера з БД. DatabaseService - клас, що містить екземпляри класів-сервісів, що забезпечують взаємодію з кожною з сутностей БД.

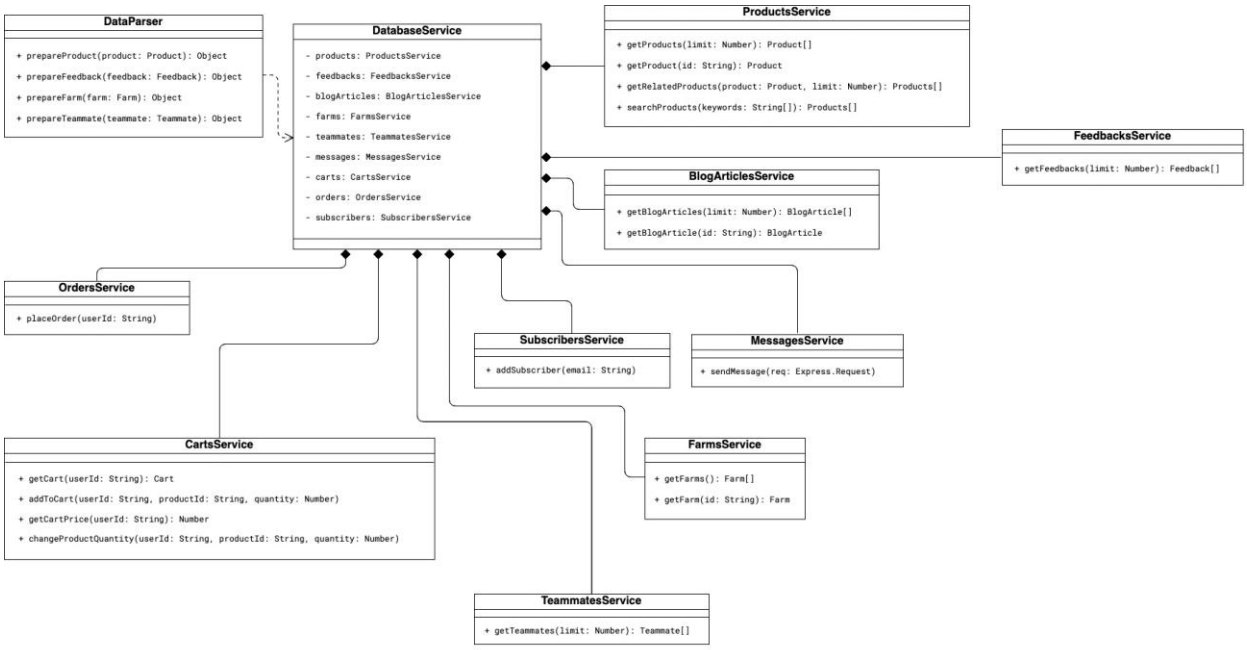


Рисунок 2.7 – Вигляд діаграми класів моделей бази даних

Клас DatabaseService реалізує JavaScript-версію патерна Singletone (об'єкт, що створюється при ініціалізації програми і існує лише в одному примірнику), що дозволяє уникнути помилок при взаємодії з БД одночасно великої кількості користувачів. Клас DataParser є допоміжним класом, що працює з результатами роботи класу DatabaseService. DataParser займається виправленням відносних шляхів, перетворенням даних про дати та приведенням їх у зрозумілий людині вигляд, а також операціями, пов'язаними з вилученням додаткових даних для відображення на вебзастосунку.

Висновки до другого розділу

В другому розділі проєкту було обрано певні технології та інструменти, які відповідають поставленим цілям і вимогам користувачів. Їхній вибір базувався на ефективності та доцільності для успішної розробки системи. Загалом, їх використання в сукупності дозволить створити потужний та ефективний вебзастосунок для продажу продуктів з інтуїтивно зрозумілим інтерфейсом та швидким пошуком на потреби користувачів.

3 РОЗРОБКА ВЕБЗАСТОСУНКУ ТА ПРОВЕДЕННЯ ТЕСТУВАННЯ

3.1 Проєктування користувацького інтерфейсу

Користувацький інтерфейс – це візуальні та інтерактивні складові цифрового продукту, наприклад, вебзастосунку або мобільного додатку, з якими користувачі взаємодіють безпосередньо. До них належать макети, кнопки, іконки та інші елементи дизайну, що спрощують навігацію і взаємодію з користувачем [24]. Дизайн, орієнтований на користувацький інтерфейс (UI), створює привабливий, зрозумілий і зручний інтерфейс, який підвищує загальний досвід користувачів. Продуманий графічний інтерфейс сприяє легкій навігації, чіткій комунікації та формує позитивне враження, що підвищує залученість і задоволеність користувачів застосунку (див. рис. 3.1).

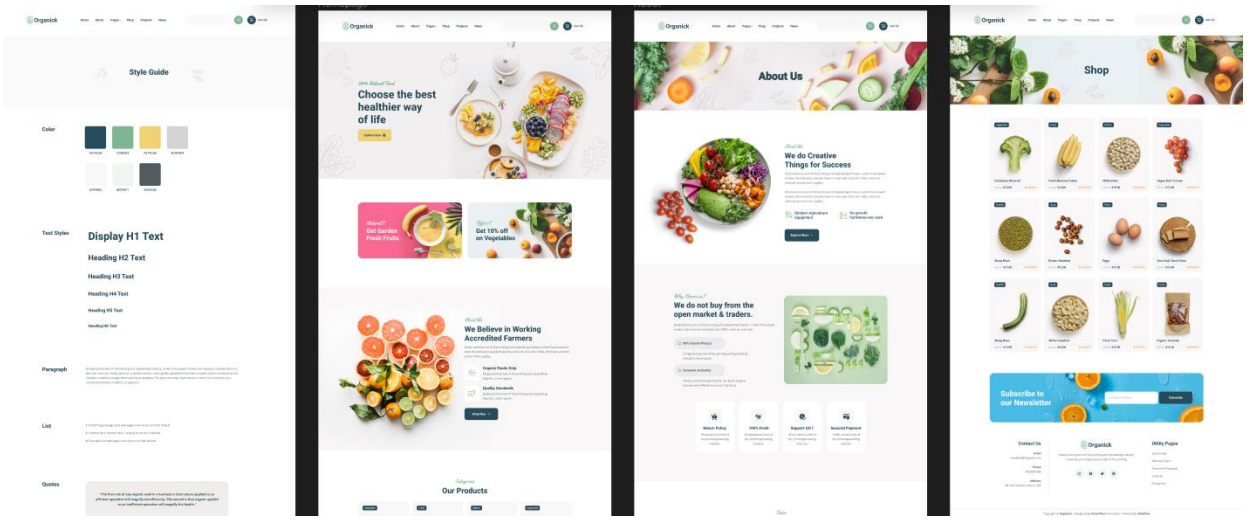


Рисунок 3.1 – Макет вебзастосунку

Для створення макету необхідні такі процеси:

– вибір фону для макет сайту;
– визначення розмірів сайту;
– робиться розмітка макету;
– робота з "шапкою" сайту;
– створення меню навігації.

3.2 Архітектура вебзастосунку

Наступним етапом у розробці будь-якого вебзастосунку є створення його архітектури. У сфері програмної інженерії поняття «архітектура» часто має широке і неоднозначне значення. Загалом, цей термін близький до поняття «проектування», особливо коли йдеться про структурний дизайн системи, а не про безпосереднє написання коду. У загальному вигляді під цим процесом варто розуміти такі основні етапи:

- поділ вебзастосунку на основні складові;
- групування складових у логічні блоки;
- визначення форматів обміну даними;
- встановлення зв’язків між компонентами;
- формування правил реалізації та розгортання;
- врахування вимог безпеки та надійності;
- оптимізація для різних типів пристроїв.

Рисунок 3.2 – Архітектура вебзастосунку

Архітектура вебзастосунку (див. рис. 3.2) передбачає наявність клієнтської частини (фронтенду), реалізованої як вебінтерфейс для користувача, що взаємодіє із серверною частиною (бекендом) через HTTP-запити (API). Серверна частина відповідає за обробку запитів, реалізацію логіки фільтрації та обробки даних, а також взаємодію з базою даних, в якій зберігаються інформація про товари та користувачів. Для зберігання та оновлення даних використовується формат CSV, який імпортується на сервер. Така архітектура забезпечує чітке розділення функціоналу відображення і обробки даних, що підвищує гнучкість, підтримуваність і масштабованість системи.

3.3 Створення сторінок вебзастосунку та його функціоналу

Кожен вебзастосунок має головну сторінку, яка називається Header (або шапка сайту) – це верхній блок на вебзастосунку. Найчастіше на ньому відображаються логотип та назва ресурсу, меню, пошук та інші елементи. В інтернет-магазинах частіше за все можна побачити форму для оформлення замовлення, пошук і корзину [25].

Шапка містить в собі:

а) Home (головна) по сторінках сайту;

б) About (про магазин);

в) Pages (сторінки) поділяється на:

1) Service (сервіс, який пропонуємо);

2) Team (команда);

3) Contact Us (контактна інформація);

г) Shop (товари, які можна купити);

д) Projects (яскрава інформація про продукт);

е) News (новини);

ж) пошук дає змогу знайти потрібний товар за назвою;

з) кошик в якому відображається обраний товар та ціну.

Логотип вебзастосунку, коли натискаєш на нього відбувається перехід до головної сторінки.

Рисунок 3.3 – Шапка вебзастосунку

Шапка вебзастосунку – це верхній елемент інтерфейсу, який відображається на всіх сторінках сайту [26]. Вона виконує роль основного навігаційного блоку, забезпечуючи швидкий доступ до ключових розділів (див. рис. 3.3).

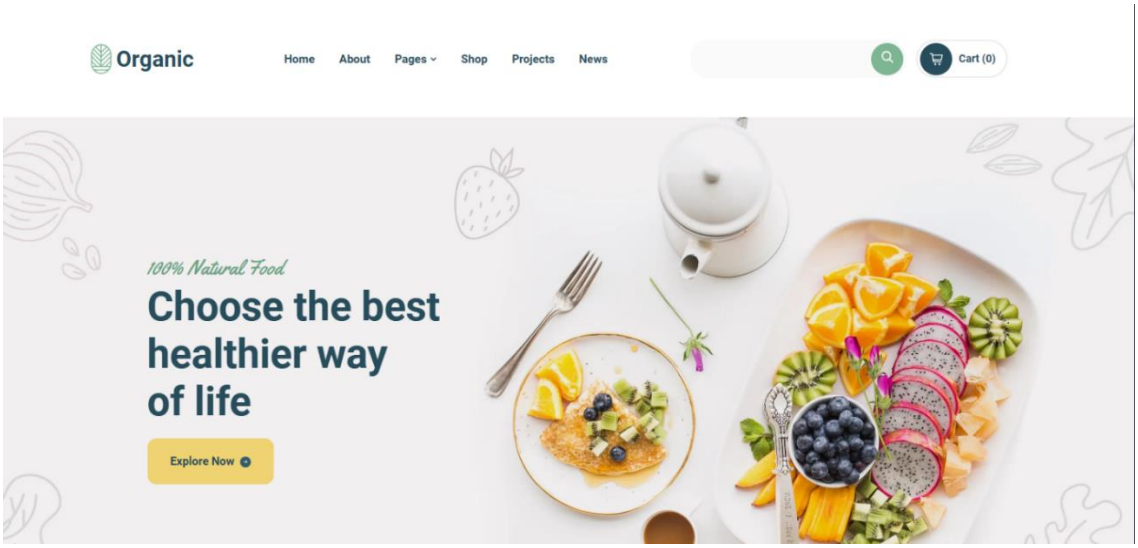


Рисунок 3.4 – Обкладинка вебзастосунку

Натиснувши на обкладинці вебзастосунку (див. рис. 3.4), можна побачити кнопку “Explore Now” (“Дослідити зараз”) (див. рис 3.5) натиснувши на неї відбувається перехід до інформації щодо магазину: звідки вони беруть продукти, до якого виду вони відносяться (в даному випадку до органічної продукції).

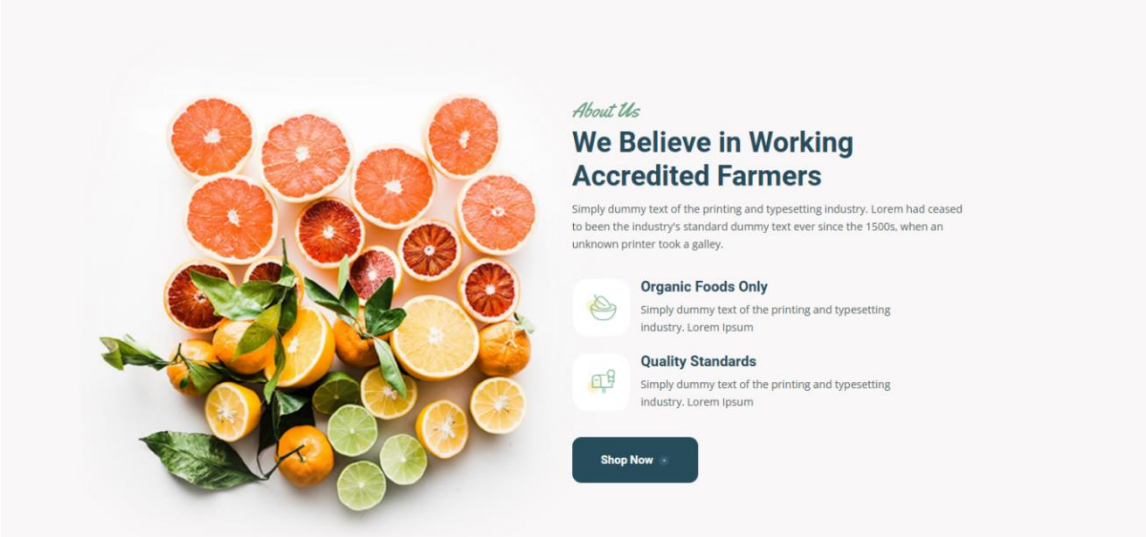


Рисунок 3.5 – Інформація про продукцію продажу

Однією з важливих частин інтернет-магазину є сторінка з контактною інформацією. Покупець може зв’язатись з представником магазину, тому що не всі

люди можуть використовувати купувати онлайн і їм легше подзвонити до магазину та замовити таким чином, або інше, наприклад, уточнення замовлення, зміна часу коли можна забрати товар чи дати свої рекомендації. Сторінка з контактною інформацією (див. рис. 3.6).

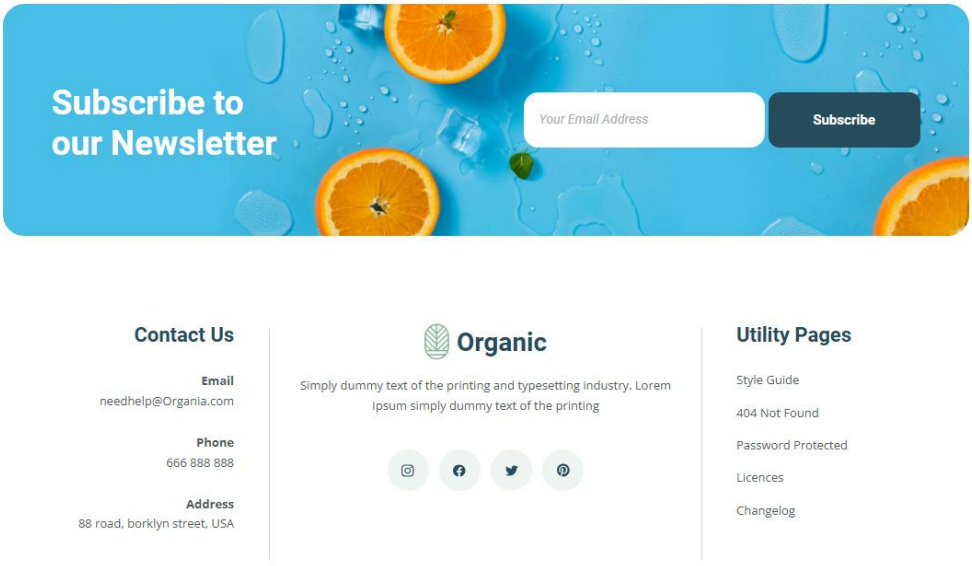


Рисунок 3.6 – Сторінка з контактною інформацією

Для більш комфортного та швидкого пошуку додаткової інформації: зв’язок із командою, сервісом та магазином (див. рис. 3.7).

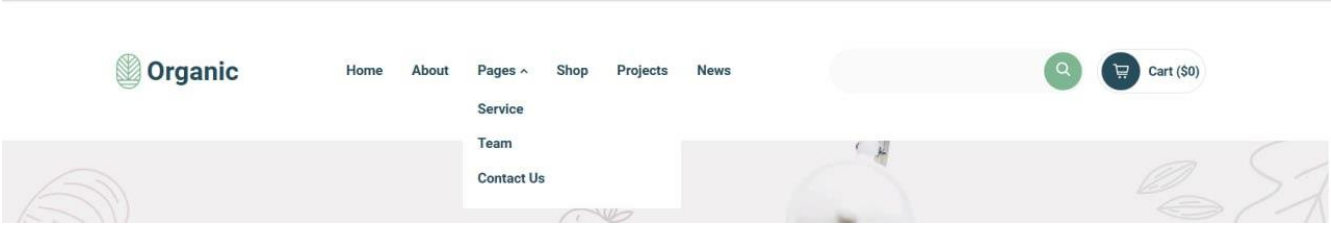


Рисунок 3.7 – Сторінка з доступними варіантами вибору

У розділі Shop (магазин) користувач має змогу переглянути весь асортимент товарів, доступних для замовлення в онлайн-магазині. Кожен товар містить детальний опис, зображення, ціну та характеристики, що допомагає зробити обдуманий вибір. Крім того, у майбутньому також з'явиться можливість

фільтрувати товари за різними параметрами для швидшого пошуку потрібного продукту. Також на сторінці реалізовано зручну структуру відображення товарів у вигляді сітки, що дозволяє швидко охопити поглядом увесь асортимент. Для кожної позиції передбачено можливість миттєвого додавання в кошик або переходу до детальнішого опису (див. рис. 3.8).

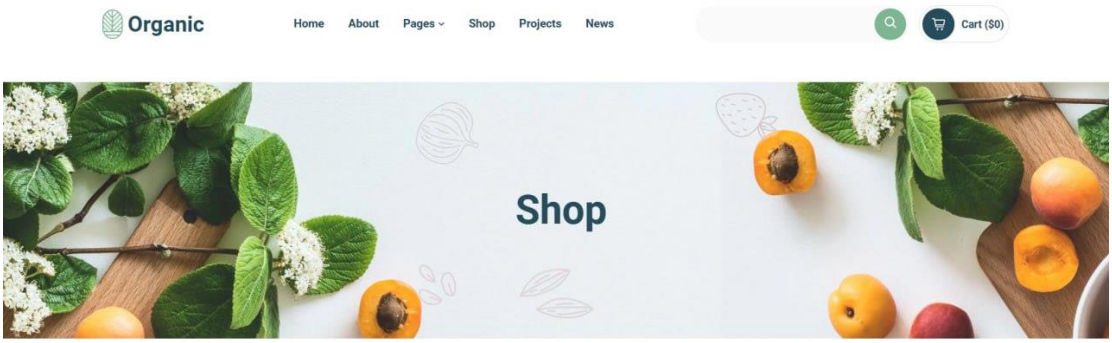


Рисунок 3.8 – Сторінка з продуктами харчування

Поки що корзина виглядає пустою, але коли в ній з’являться продукти, тут відобразиться весь їх список разом із загальною вартістю, що дозволить легко керувати покупками перед оформленням замовлення (див. рис. 3.9).

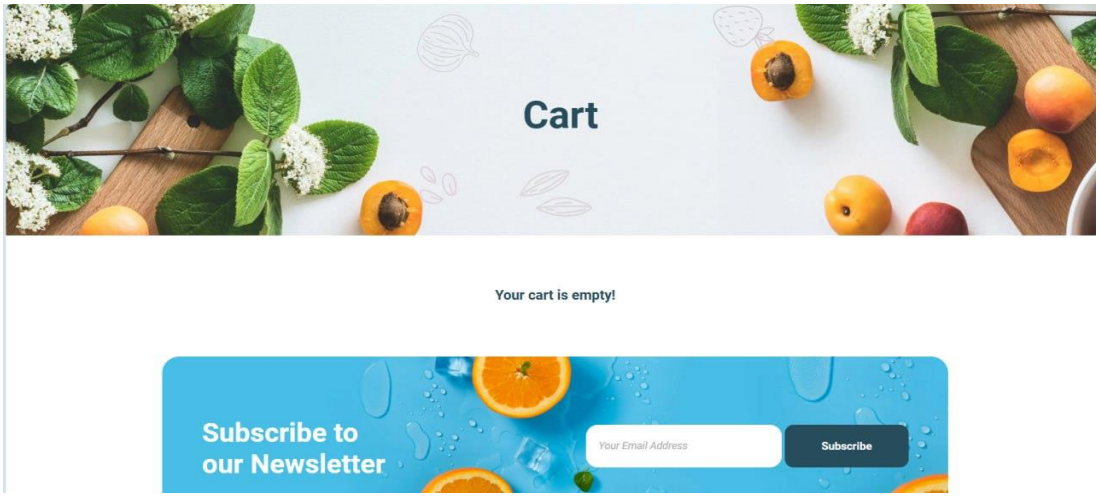


Рисунок 3.9 – Корзинка

Місце для зворотнього зв’язку – це розділ онлайн-магазину, створений спеціально для прямого та відкритого спілкування з клієнтами [27]. Тут можна поділитися враженнями, досвідом, побажаннями або зауваженнями після купівлі. Ця функція дозволяє бути в курсі проблем та швидко виправляти їх, покращувати якість сервісу (див. рис 3.10).

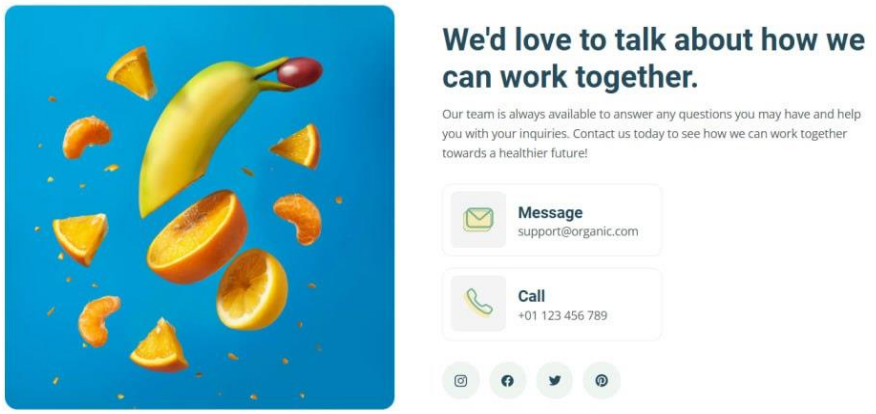


Рисунок 3.10 – Інформація про роботу вебзастосунку

Також воно потрібне, щоб команда, яка отримає повідомлення від клієнтів могла проаналізувати і покращити роботу онлайн-магазину, розуміючи потреби користувачів, оновлюючи асортимент, уточнюючи навігацію та поліпшуючи обслуговування. Ця опція працює цілодобово, тому покупець може залишити своє звернення у зручний час. Такий підхід сприяє швидкому реагуванню на проблеми та забезпеченню стабільної роботи вебзастосунку. Залучення клієнтів до комунікації створює враження відкритості й турботи з боку компанії. Аналіз звернень допомагає виявити популярні товари та визначити тенденції попиту. Тому це дозволяє оперативно адаптувати пропозицію магазину під актуальні потреби ринку. Загалом, функція зворотного зв’язку є важливою складовою якісного онлайн-сервісу. Це підвищує рівень довіри та залученості клієнтів, а також формує позитивне враження від взаємодії або роботи з сервісом (див. рис 3.11).

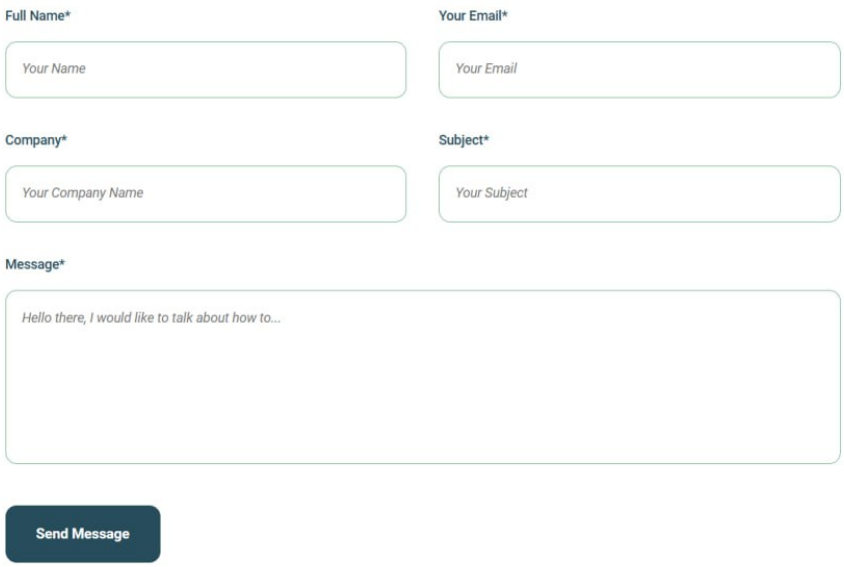


Рисунок 3.11 – Місце для зворотного зв’язку

У кожному онлайн-магазині є розділ, де працівники можуть написати: часті питання від покупців, корисні статті чи блоги, політику конфіденційності та умови використання вебзастосунку. Тому він призначений для детального ознайомлення користувачів із продукцією ферми (див. рис. 3.12).

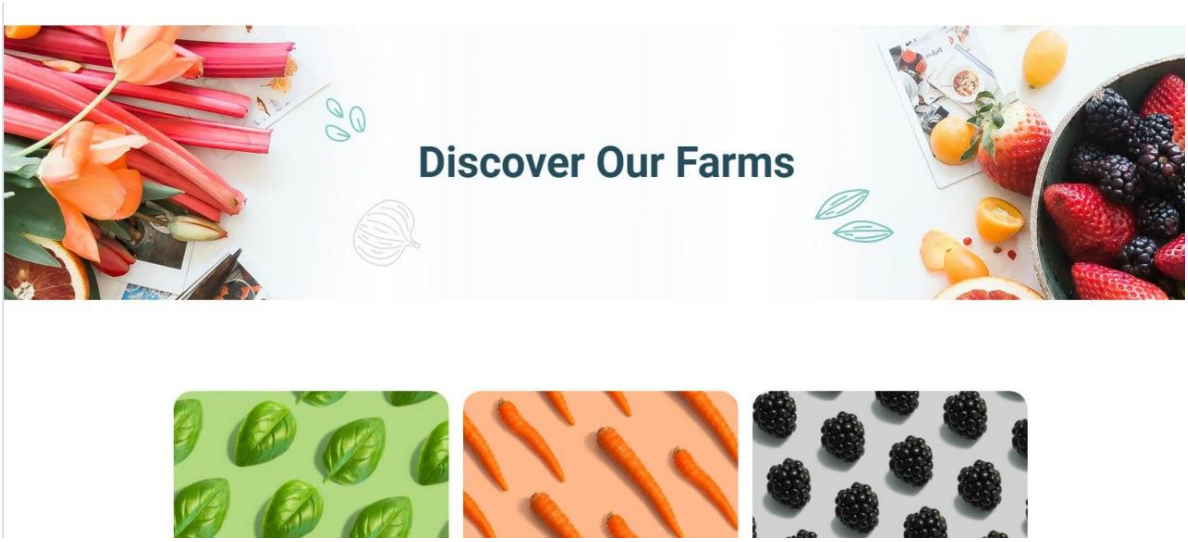


Рисунок 3.12 – Інформаційний розділ

Тут міститься відомості про походження продуктів, періоди їх збору, основні характеристики та категорії, до яких вони належать. Це також формує позитивний

імідж, показуючи звідки поступають товари і підтверджуючи дотримання принципів екологічності та натуральності продукції, яка потім потрапить в магазин. (див. рис. 3.13 – 3.14).

Рисунок 3.13 – Інформація про продукт

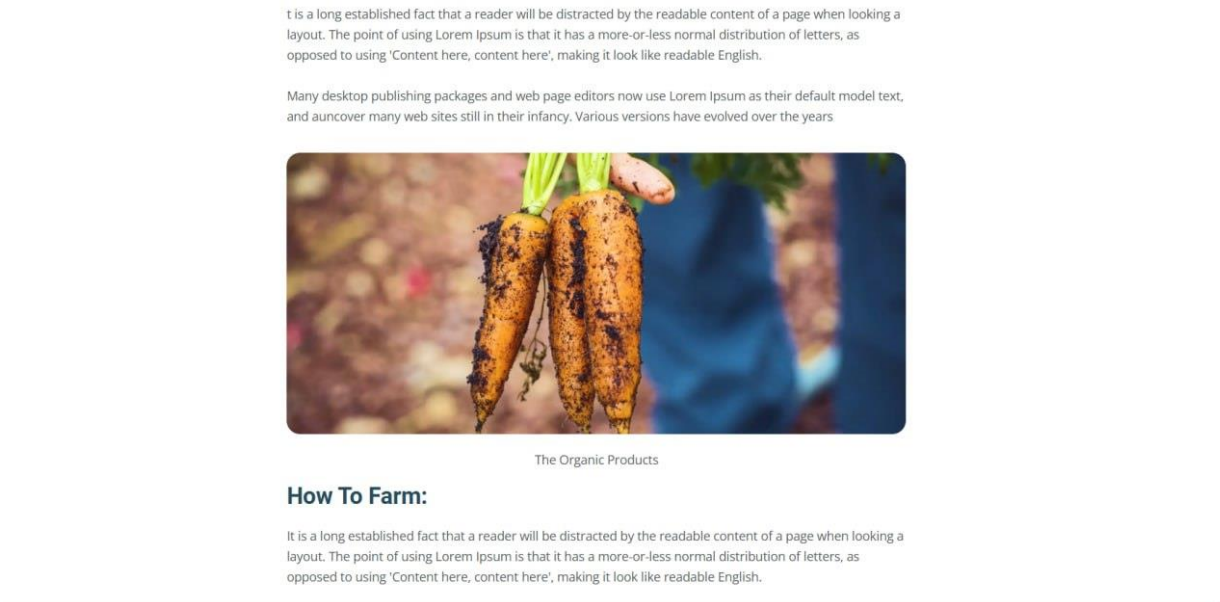


Рисунок 3.14 – Опис продукту

У результаті створення сторінок вебзастосунку було реалізовано основний функціонал, який забезпечує зручну навігацію та зрозумілу взаємодію користувачів із онлайн-магазином. Кожна сторінка виконує поставлені завдання:

від шапки, до зворотного зв'язку або представленні інформації про продукти. Розроблена структура забезпечує простоту масштабування додатку та полегшує впровадження нових функцій у подальшому. Загалом, реалізація функціоналу сторінок забезпечує комфортний та ефективний досвід користувачів, що є важливою складовою успішного вебзастосунку.

3.4 Адаптивність вебзастосунку

Адаптивність вебзастосунку – це правильний і зручний дизайн застосунку, який демонструється користувачу, коли він заходить на сайт зі свого пристрою, незалежно від розміру екрану [28]. Цей тип дизайну використовується для автоматичного налаштування розміру сайту відповідно до вікна браузера (див. рис. 3.15).

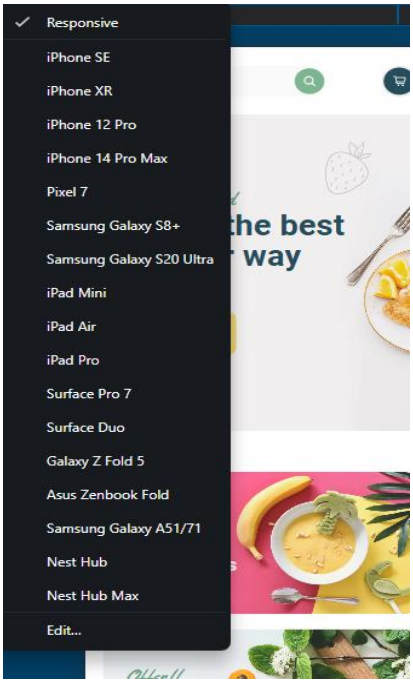


Рисунок 3.15 – Адаптивність сторінки під різни пристрої

Сьогодні користувачі мають широкий спектр пристроїв: від моделей iPhone SE та iPhone 12 Pro до Pixel 7, Samsung Galaxy S8+ і S20 Ultra, різних моделей iPad Mini, iPad Air, iPad Pro, Surface Pro, гнучких смартфонів, Galaxy Z Fold 5 і Asus

ZenBook Fold, а також відносно бюджетних пристроїв Samsung A51/71 та розумних дисплеїв Nest Hub і Nest Hub Max (див. рис. 3.16 – 3.19).

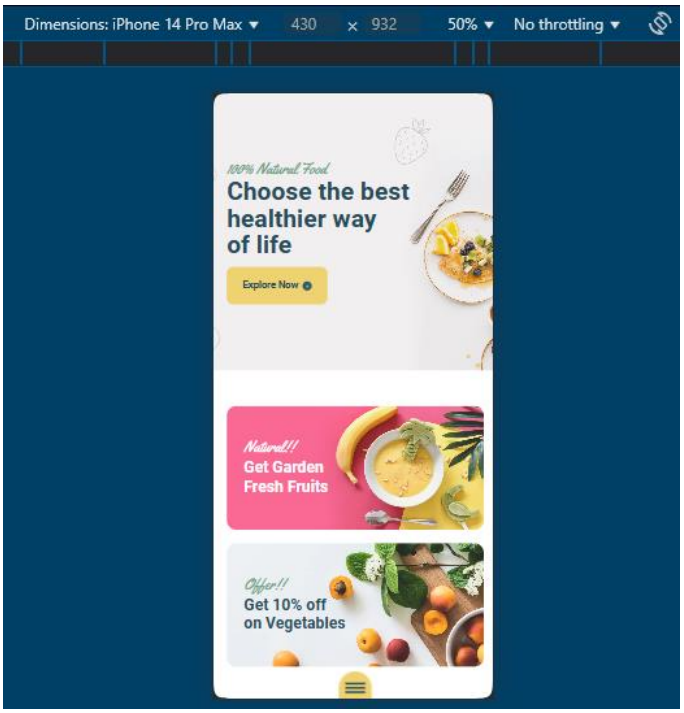


Рисунок 3.16 – Адаптивність сторінки під iPhone 14 Pro Max

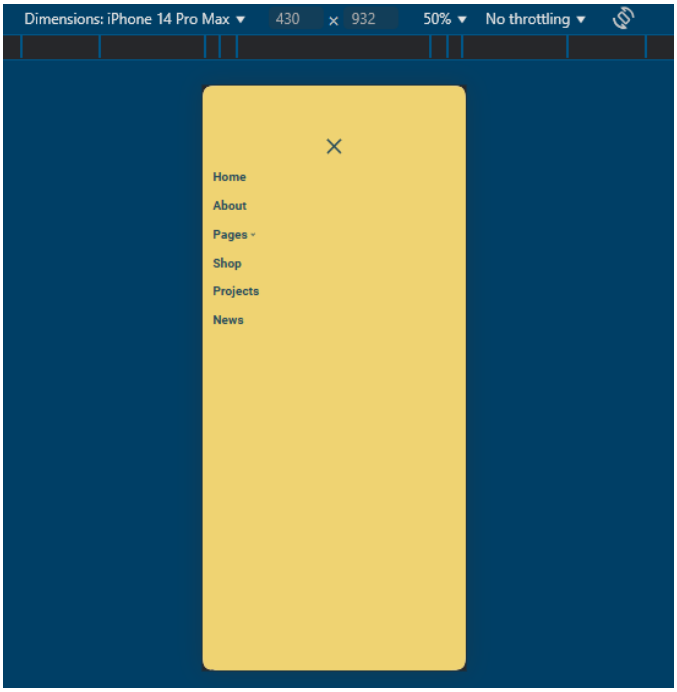


Рисунок 3.17 – Меню навігацій iPhone 14 Pro Max

На телефонах меню навігації знаходиться знизу: з трьома горизонтальними лініями. Воно розміщується знизу посередині екрана. Коли користувач проводить пальцем вгору або натискає на цю іконку, відкривається повне меню з усіма розділами вебзастосунку. Це зроблено для зручності та щоб не займати багато місця на екрані, але при цьому мати швидкий доступ до всіх сторінок, не перевантажуючи інтерфейс.

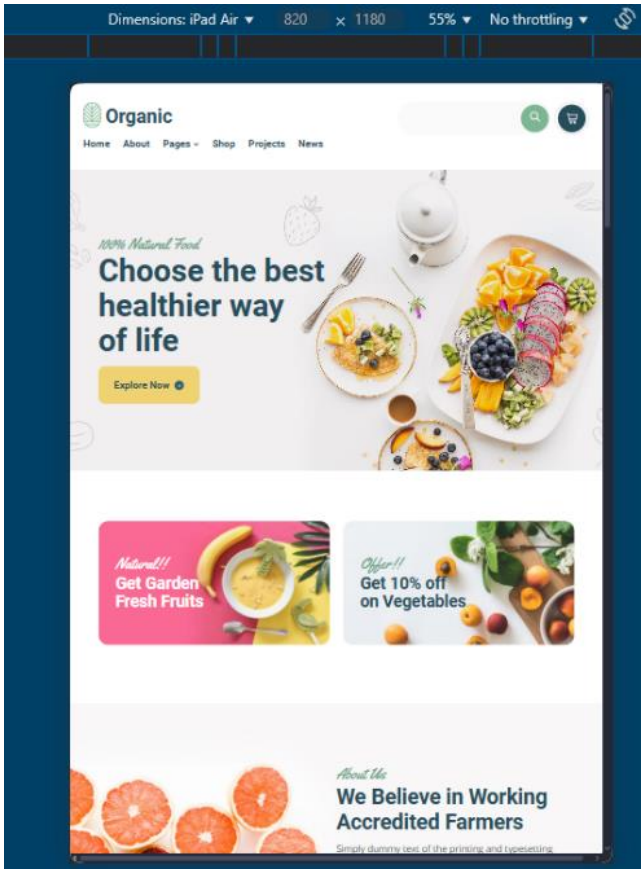


Рисунок 3.18 – Адаптивність сторінки під iPad Air

Планшети займають проміжне місце між телефонами, ноутбуками. Тому адаптація під планшети враховує значення як більший екран, ніж у телефонів, але менший і з іншою взаємодією, ніж у ноутбуків. Від ноутбуків планшети відрізняються, зокрема, сенсорним управлінням і часто вертикальною орієнтацією екрану, а від телефонів більшим простором для розміщення контенту. Тому адаптація під планшет не зводиться повністю до версій для телефонів чи ноутбуків.

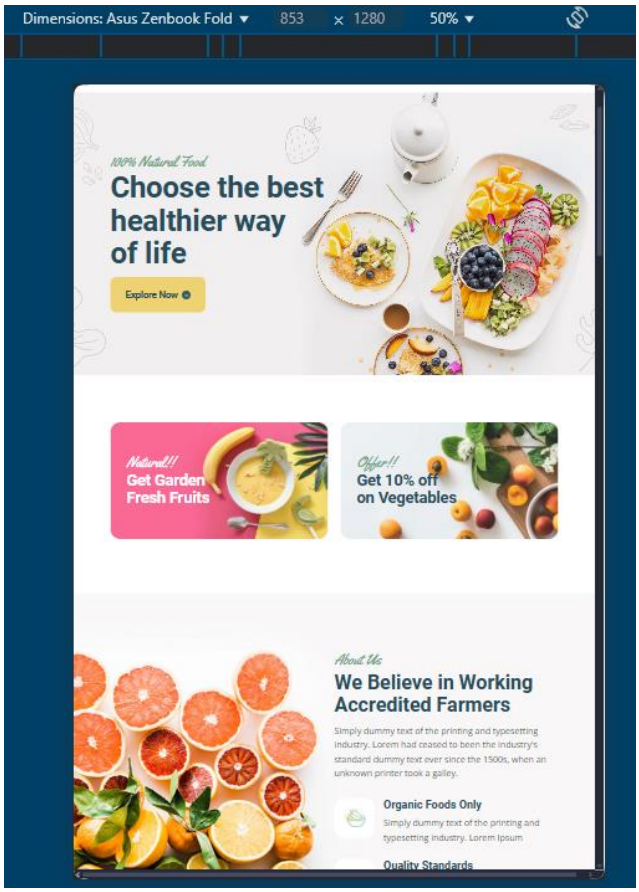


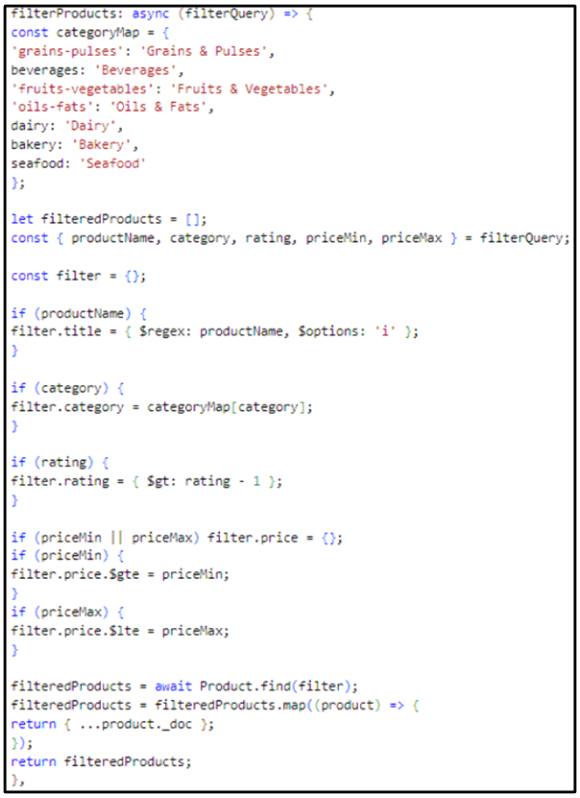
Рисунок 3.19 – Адаптивність сторінки під Asus ZenBook Fold

Ці пристрої суттєво різняться як за розмірами і роздільною здатністю екранів, так і за операційними системами. Відтак вебзастосунок повинен забезпечувати коректне відображення і зручність використання на будь-якому з цих пристроїв. Це є важливою умовою для охоплення максимально широкої аудиторії та створення комфортного користувацького досвіду незалежно від платформи чи технічних характеристик гаджета.

3.5 Вирішення проблеми великих даних

У вебзастосунку продажу продуктів харчування питання ефективної обробки великих даних є важливим фактором забезпечення стабільної роботи системи та зручності для користувача. Реалізовані механізми обробки даних дозволяють підтримувати ефективність системи при зростанні кількості товарів. Для цього у вебзастосунку реалізовано обробку даних, зокрема фільтрацію товарів за

ключовими параметрами: ціною, категорією та рейтингом. Такий підхід сприяє продуктивній роботі з великими обсягами даних, забезпечуючи користувачеві швидкий доступ до найбільш релевантних товарів. Завдяки структурованій організації даних та оптимізованим запитам, система підтримує стабільну продуктивність навіть за умови розширення асортименту.



```
filterProducts: async (filterQuery) => {
const categoryMap = {
'grains-pulses': 'Grains & Pulses',
beverages: 'Beverages',
'fruits-vegetables': 'Fruits & Vegetables',
'oils-fats': 'Oils & Fats',
dairy: 'Dairy',
bakery: 'Bakery',
seafood: 'Seafood'
};

let filteredProducts = [];
const { productName, category, rating, priceMin, priceMax } = filterQuery;

const filter = {};

if (productName) {
filter.title = { $regex: productName, $options: 'i' };
}

if (category) {
filter.category = categoryMap[category];
}

if (rating) {
filter.rating = { $gt: rating - 1 };
}

if (priceMin || priceMax) filter.price = {};
if (priceMin) {
filter.price.$gte = priceMin;
}
if (priceMax) {
filter.price.$lte = priceMax;
}

filteredProducts = await Product.find(filter);
filteredProducts = filteredProducts.map((product) => {
return { ...product._doc };
});
return filteredProducts;
},
```

Рисунок 3.20 – Фрагмент коду фільтрації

Цей код (див. рис. 3.20) дозволяє сортувати продукти у БД за різними параметрами. Спочатку створюється об'єкт categoryMap, який є звичайним словником, що допомагає зіставити значення категорій у HTML-формі зі своїми назвами категорій. Таке рішення необхідне, тому що в коді HTML пробіли або спеціальні символи у значеннях опцій лише дефіси. Таким чином, ми можемо привести вибране значення у потрібний формат за допомогою цього словника. Найменування товару, категорія, рейтинг та діапазон цін є параметрами фільтрації, отриманими з об'єкта filterQuery. Тобто, якщо назва продукту вказана, додається

умова, за якою шукаються всі товари у назві яких зустрічається це слово або його частина, незалежно від написання (реєстр не враховується). Якщо обирається категорію, тоді перекладається через categoryMap і теж додається до фільтру або коли вказано рейтинг – додається відповідна умова. Те саме з ціною: мінімальне та/або максимальне значення, вони також враховуються. Після збору всіх умов використовується Product.find() для пошуку продуктів в БД, а потім повертається список продуктів, відповідних заданим параметрам (див. рис. 3.21 – 3.24).

Для реалізації фільтрації товарів у вебзастосунку було підключено зовнішній Dataset із інформацією про продукти харчування. Спочатку дані з Dataset були додані до проєкту у форматі CSV файлу, що дозволяло зручно їх обробляти та фільтрувати. Зараз ці дані підтягуються безпосередньо з бази даних, що значно спрощує доступ до актуальної інформації та робить процес більш автоматизованим.

Для фільтрації використовуються метадані про продукти. Наприклад, категорія, ціна, назва, рейтинг тощо. Ці дані можуть використовуватися, аби згенерувати для користувача рекомендації швидше, ніж він зробить таку кількість дій на платформі, що необхідна для роботи алгоритмів фільтрації та побудови матриці користувач-продукт. Для цього використовується алгоритм, який послідовно відбирає товари, що відповідають критеріям користувача. Використання структурованих метаданих дозволяє мінімізувати час обробки запитів та забезпечити актуальні результати навіть при збільшенні обсягу даних. Це підвищує загальну ефективність системи і сприяє більш персоналізованому та зручному у користуванні.

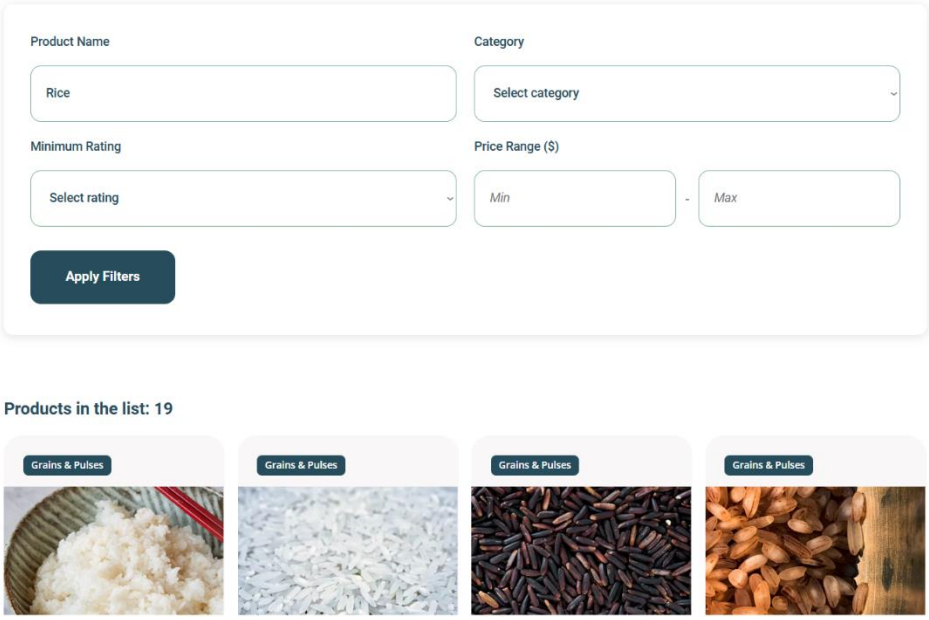


Рисунок 3.21 – Фільтрація за назвою продукту

Фільтрація за назвою продукту відбувається шляхом введення користувачем назви продукту в пошукове поле. Система порівнює запит із назвами товарів у БД та відображає лише відповідні результати.

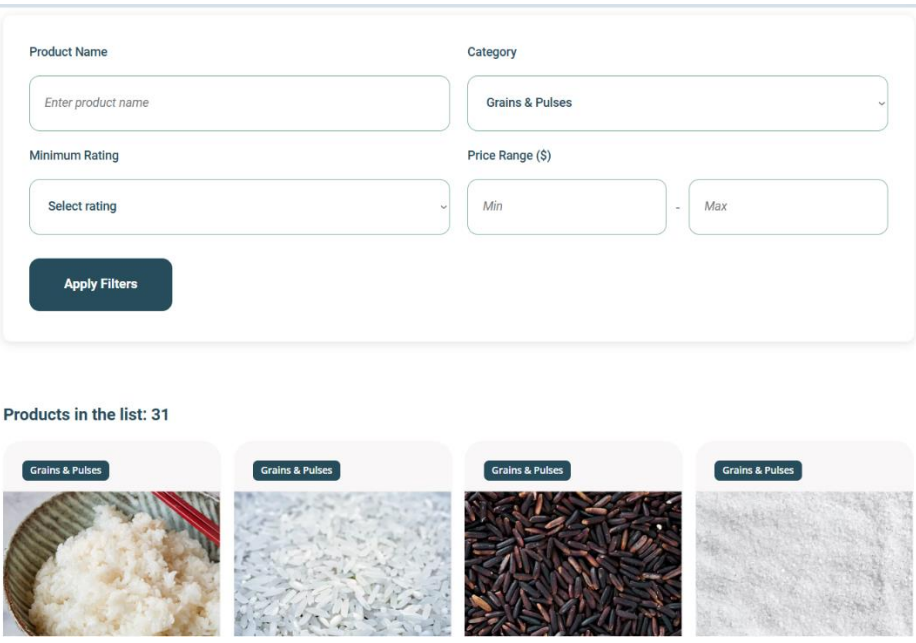


Рисунок 3.22 – Фільтрація за категорією

Фільтрація за категорією, коли користувач обирає бажану категорію з доступного списку фільтрів. Після цього система автоматично відображає лише ті товари, що належать до обраної категорії.

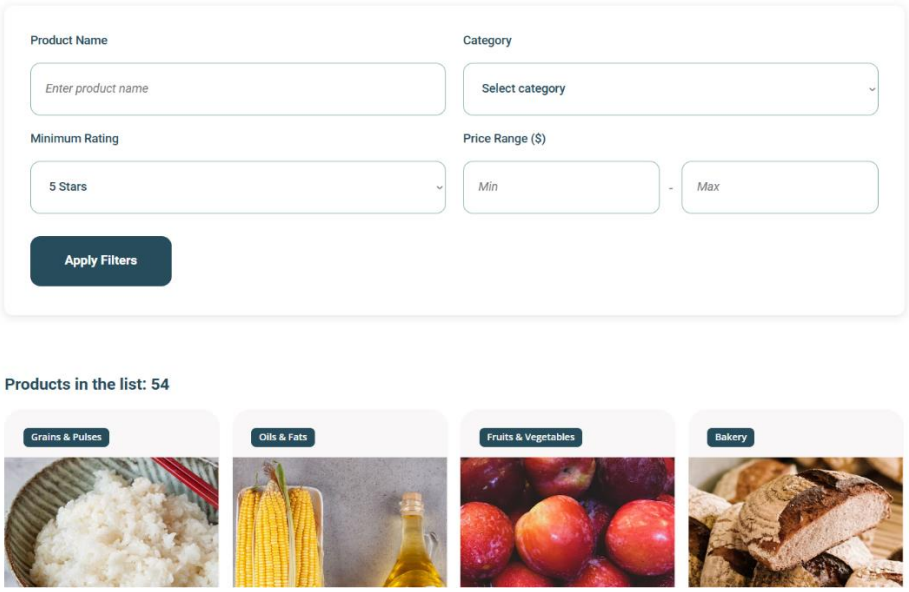


Рисунок 3.23 – Фільтрація за рейтингом

Фільтрація за рейтингом. Користувач може впорядкувати товари обравши відповідний параметр зі списку сортування. Тоді відображається продукти у порядку від найвищого до найнижчого рейтингу або навпаки, залежно від вибору.

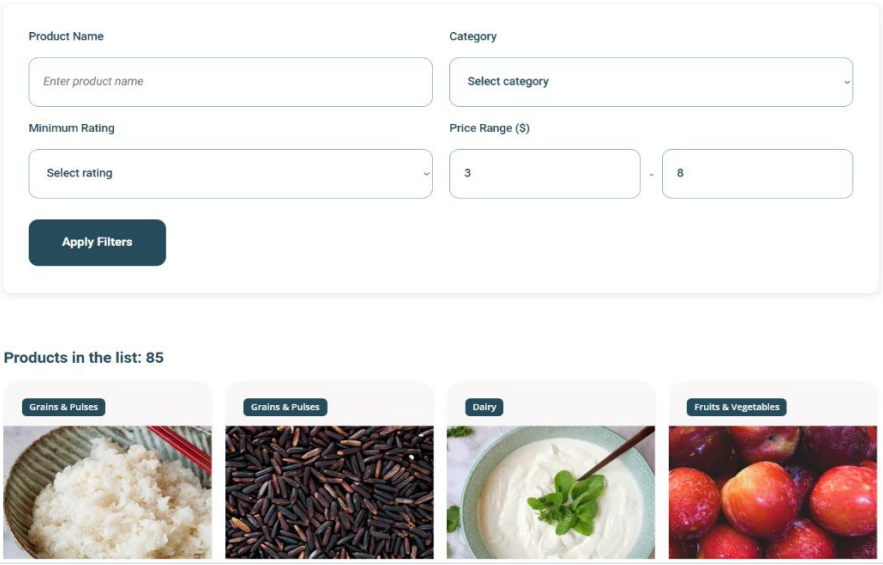


Рисунок 3.24 – Фільтрація за ціною

Фільтрація за ціною дозволяє користувачу задати бажаний ціновий діапазон за допомогою повзунка або введення значень вручну. Система автоматично оновлює список товарів, відображаючи лише ті, що відповідають зазначеним межам.

3.6 Проєктування БД

БД широко застосовуються для динамічних застосунків із великими обсягами інформації – це можуть бути інтернет-магазини, портали або корпоративні ресурси. Таблиця в БД представляє собою масив однорідних елементів, які називаються записами. Кожен запис може містити один або кілька іменованих полів. Назви та порядкові номери полів визначаються під час створення таблиці, а кожне поле має свій тип даних.

Дані зберігаються у форматі документів або колекцій. Розробники вбачають цей продукт як проміжне рішення між традиційними реляційними СУБД та NoSQL системами. На відміну від реляційних баз даних, MongoDB не застосовує жорсткі схеми, що сприяє підвищенню загальної ефективності роботи системи.

Опис БД, (див. в табл. 3.1 – 3.9).

Таблиця 3.1 – Опис бази даних, колекція blogarticles

Назви полей	**Тип**	**Збереження інформації**
_id	ObjectId	Унікальний ключ
Date	Date	Дата створення статті
Author	String	Ім’я автора статті
Title	String	Назва статті
Description	String	Короткий опис статті
Content	String	Наповнення статті
promoImageNameMini	String	Шлях до мініатюри зображення у шапці статті на сервері
promoImageName	String	Шлях до зображення у шапці статті на сервері

Таблиця 3.2 – Опис бази даних, колекція carts

Назви полей	Тип	Збереження інформації
_id	ObjectId	Унікальний ключ
userId	String	Локальний id користувача
Filling	Object	Наповнення корзини, зберігає пари: product._id, кількість товару

Таблиця 3.3 – Опис бази даних, колекція farms

Назви полей	Тип	Збереження інформації
_id	ObjectId	Унікальний ключ
Name	String	Назва ферми
Type	String	Тип ферми
promoImageName	String	Шлях до зображення у шапці статті на сервері
Category	String	Категорія ферми
clientName	String	Ім’я замовника та володаря ферми
Content	String	Наповнення статті з інформацією про ферму
Date	Date	Дата заснування ферми
Description	String	Короткий опис ферми
Service	String	Тип продуктів, ввирощуваних на фермі
promoImageNameMini	String	Шлях до мініатюри зображення у шапці статті на сервері

Таблиця 3.4 – Опис бази даних, колекція feedbacks

Назви полей	Тип	Збереження інформації
_id	ObjectId	Унікальний ключ
Name	String	Ім’я клієнта
Status	String	Положення клієнта стосовно компанії
Rating	Int32	Оцінка, яку дав клієнт сервісу компанії
feedbackText	String	Думка клієнта про компанію, якою він поділився
avatarName	String	Шлях до фотографії клієнта на сервері

Таблиця 3.5 – Опис бази даних, колекція messages

Назви полей	Тип	Збереження інформації
_id	ObjectId	Унікальний ключ
fullName	String	Повне ім’я людини

Кінець таблиці 3.5

Назви полей	**Тип**	**Збереження інформації**
Email	String	Адреса електронної пошти людини
Company	String	Назва компанії, яку представляє людина
Subject	String	Тема звернення
Message	Int32	Текст повідомлення

Таблиця 3.6 – Опис бази даних, колекція orders

Назви полей	**Тип**	**Збереження інформації**
_id	ObjectId	Унікальний ключ
userId	String	Локальний id користувача
Filling	Object	Наповнення корзини, зберігає пари: product._id, кількість товару
Price	Int32	Комулятивна ціна усього товару у кошику

Таблиця 3.7 – Опис бази даних, колекція products

Назви полей	**Тип**	**Збереження інформації**
_id	ObjectId	Унікальний ключ
Category	String	Назва категорії продукту
Title	String	Назва продукту
oldPrice	Double	Стара ціна на продукт
Price	Double	Поточна ціна на продукт
imgName	String	Шлях до зображення продукту на сервері
Rating	Int32	Рейтинг продукту
additionalInfo	String	Додаткова інформація про продукт
Description	String	Повний опис продукту
promoInfo	String	Короткий опис продукту

Таблиця 3.8 – Опис бази даних, колекція subscribers

Назви полей	**Тип**	**Збереження інформації**
_id	ObjectId	Унікальний ключ
Email	String	Поштова адреса людини, яка побажала отримувати новинну розсилку

Таблиця 3.9 – Опис бази даних, колекція teammates

Назви полей	**Тип**	**Збереження інформації**
_id	ObjectId	Унікальний ключ

Кінець таблиці 3.9

Назви полей	Тип	Збереження інформації
Name	String	Ім’я члену команди працівників компанії
facebookLink	String	Адреса посилання на Facebook-сторінку члену команди працівників компанії
Occupation	String	Професія члену команди працівників компанії
photoName	String	Шлях до фото члену команди працівників компанії на сервері
twiterLink	String	Адреса посилання на Twitter-сторінку члену команди працівників компанії
instagramLink	String	Адреса посилання на Instagram-сторінку члену команди працівників компанії

Переваги та недоліки, (див. в табл. 3.10).

Таблиця 3.10 – Переваги та недоліки MongoDB

Переваги	Недоліки
Розроблено на мові програмування C++, легко інтегрувати в будь-яку операційну систему	Ця база даних не настільки сумісна з вимогами ACID (атомність, узгодженість, ізоляція та довговічність), як реляційні бази даних
Можна доставити кінцевому клієнту як хмарне рішення	Транзакції з використанням MongoDB є складними
Технологія застосовується до будь-якого поля документа на розсуд користувача. Проіндексована інформація обробляється швидше	MongoDB не має засобів для збережених процедур або функцій
Використовує колекції замість таблиць. Вони містять різні типи наборів даних	Значне навантаження на пам’ять при обробці великих обсягів даних

Отже, після проєктування БД та її підключення до проєкту забезпечено коректну взаємодію між вебзастосунком і сховищем даних. Це дозволяє ефективно зчитувати, обробляти та зберігати інформацію, необхідну для роботи онлайн-магазину.

3.7 Тестування вебзастосунку

Тестування вебзастосунку – це технічний процес технічний процес

дослідження, спрямований на визначення якості продукції з урахуванням умов її експлуатації [29]. Методи тестування також охоплюють виявлення помилок та інших дефектів, а також перевірку функціональності програмних компонентів для їх оцінки.

Оцінка може включати такі аспекти:

– відповідність вимогам, які встановлені для розробників та проектувальників;

– правильність обробки усіх можливих варіантів вхідних даних;

– дотримання часових рамок, визначених для виконання функцій;

– сумісність із різними операційними системами.

Тестування відбувалось, як тільки весь код був виконуваний. Процес розробки зазвичай включає визначення часу та умов проведення тестування. Наприклад, якщо розробка здійснюється поетапно, більшість тестів проводяться після затвердження системних вимог і лише після цього реалізуються у вигляді тестових програм. Перш за все необхідно заповнити всі форми та перевірити, як дані відображаються в БД. Далі слід скористатися пошуком, щоб знайти потрібний продукт харчування та подивитись чи зчитується вся інформація про нього, яка є в БД. Наступним етапом тестування є додавання обраного продукту до кошика з подальшою перевіркою зчитування інформації БД. Також треба перевірити видалення та зміни кількості товару. Після цього потрібно перейти до кошика та заповнити форму оформлення замовлення (див. рис. 3.25 – 3.31).

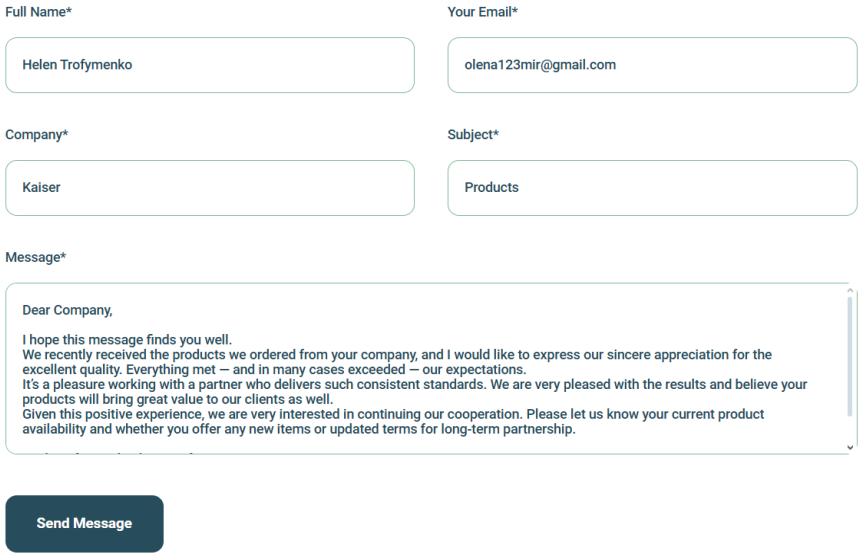


Рисунок 3.25 – Заповнення зворотного зв'язку

Заповнивши форму переходимо до БД, щоб перевірити коректність збереження введених даних (див. рис. 3.26).

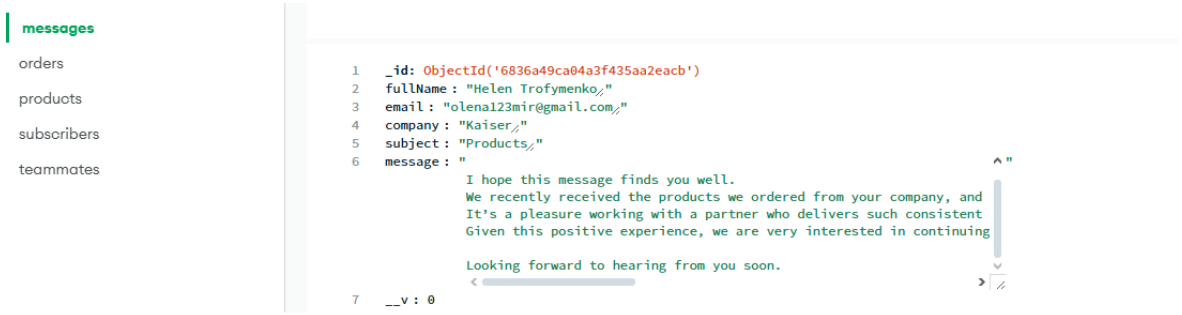


Рисунок 3.26 – Перевірка в БД заповненого

Перевіривши коректність збережених даних у БД, наступним етапом є тестування функції пошуку товару в магазині (див. рис. 3.27).

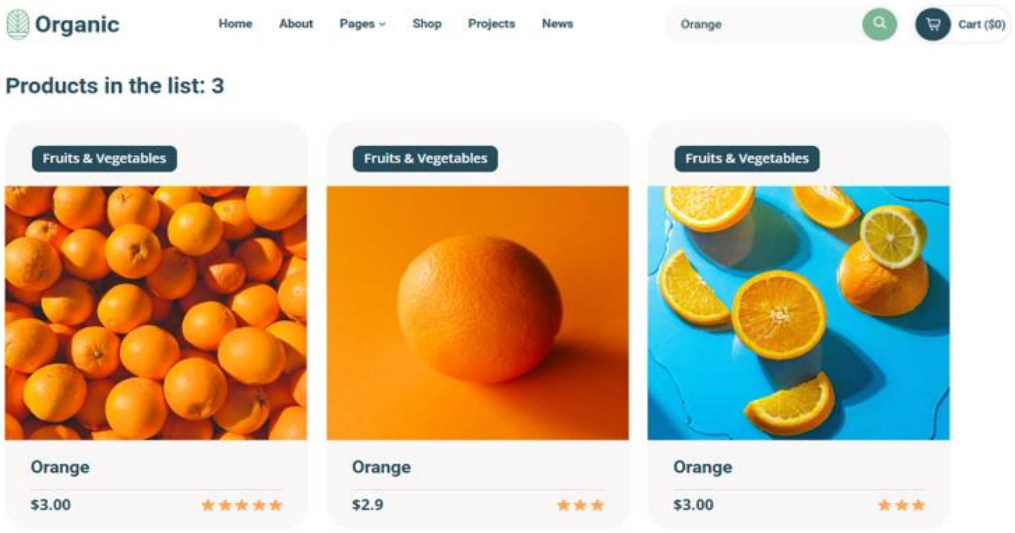


Рисунок 3.27 – Пошук продукта

Переконавшись, що товар відображається в результатах пошуку, обираємо продукт для додавання до кошика (див. рис. 3.28).

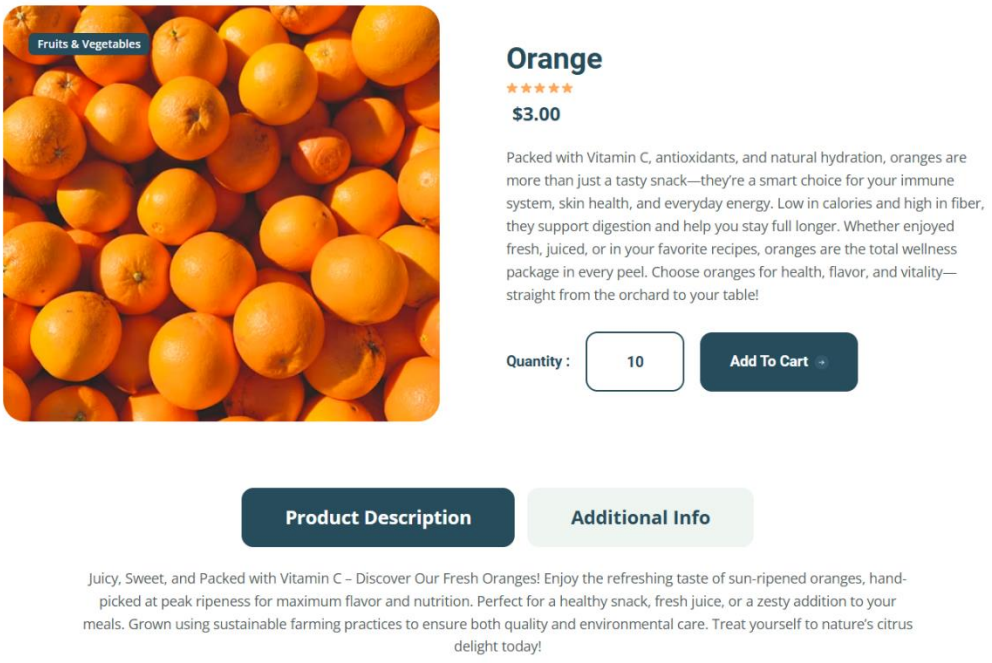


Рисунок 3.28 – Вибір продукта

Після додавання обраного продукту до кошика переходимо до перегляду вмісту кошика для подальшого оформлення замовлення (див. рис. 3.29).

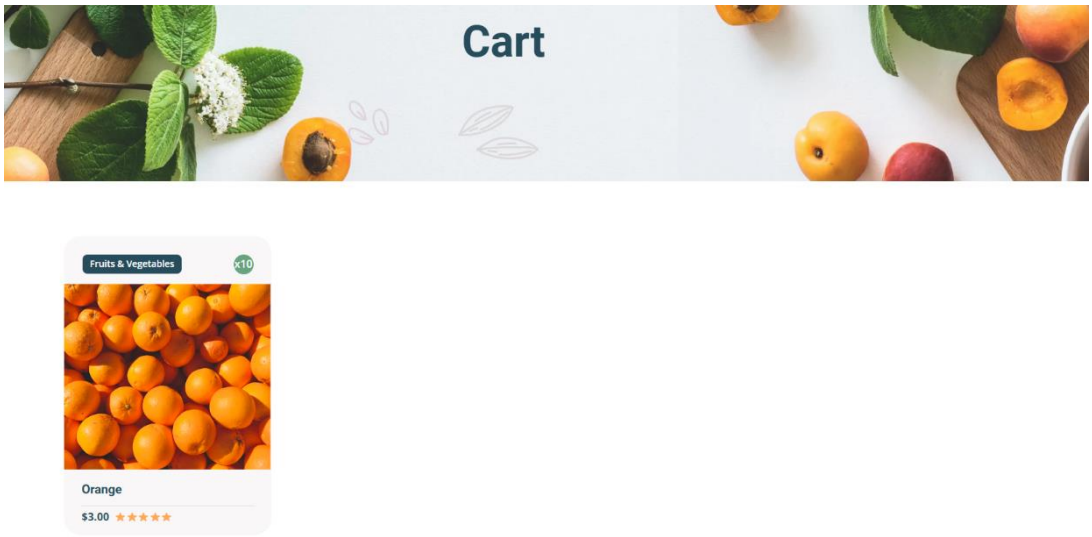


Рисунок 3.29 – Перехід на сторінку корзини покупок

Коли було додано продукти до кошика є можливість коригувати кількість та видаляти обрані позиції (див. рис. 3.30).

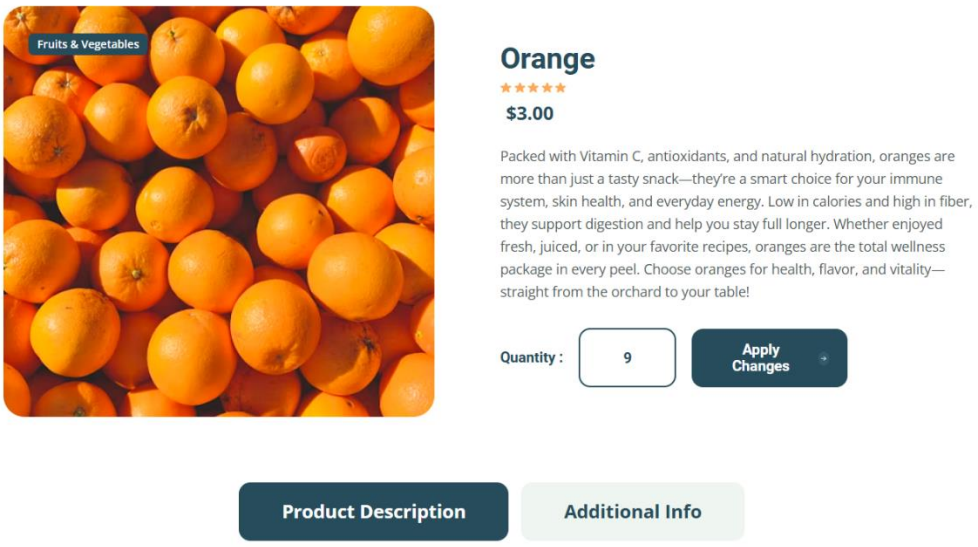


Рисунок 3.30 – Перевірка роботи кошика

Останнім кроком є оформлення замовлення, тобто введення необхідної інформації , для остаточного підтвердження обраних продуктів харчування (див. рис. 3.31).

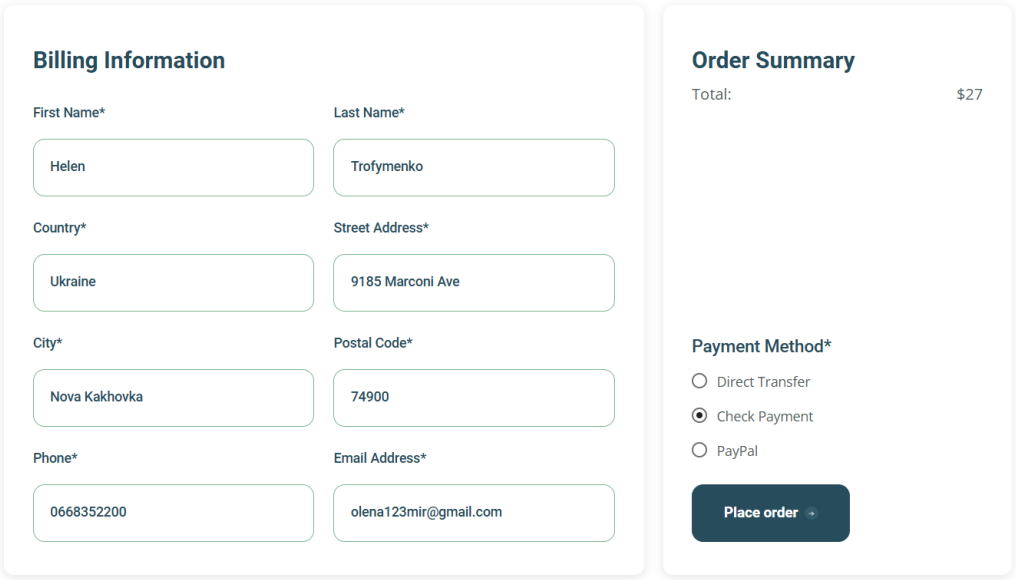


Рисунок 3.31 – Оформлення замовлення

Рисунок 3.32 – Замовлення успішно прийнято

Необхідно впевнитися, що після успішного підтвердження оплати замовлення коректно з’явились в БД (див. рис. 3.33).

Рисунок 3.33 – Перевірка замовлення в БД

Перевірка показала, що вебзастосунок може використовуватись користувачем, має зручний інтерфейс, навігаційну панель та зрозумілу логічну структуру. Під час перевірки не виявлено помилок в коді.

3.8 Інструкції користувача вебзастосунку

В цьому розділі наведено інструкцію використання вебзастосунку, що дозволяє користувачеві здійснити повноцінне замовлення товарів та отримати необхідну інформацію:

1) **навігація каталогом продуктів:** після переходу на головну сторінку вебзастосунку користувачеві надається доступ до повного переліку продуктів харчування. Для зручності передбачено можливість пошуку за ключовими словами, а також використання системи фільтрації за категоріями, рейтингом, ціною тощо;

2) **перегляд інформації про продукт:** для перегляду інформації щодо продукту, необхідно натиснути на зображення або його назву, тоді відбувається перехід на більш детальну інформацію, а саме те, що там відображаються такі дані, як опис, ціна, склад, категорія та інші параметри.

3) **формування кошика:** на сторінці обраного товару слід зазначити бажану кількість та натиснути кнопку «Додати до кошика». Цю процедуру також можна повторити для інших продуктів.

4) **оформлення замовлення:** після завершення вибору необхідно перейти до розділу «Кошик», перевірити список обраних товарів, а також за потреби змінити кількість або видалити окремі позиції. Далі слід заповнити форму із зазначенням контактної інформації, куди робиться доставка та вибрати оплату.

5) **звернення до служби підтримки або написання зворотнього повідомлення:** у разі виникнення технічних або організаційних питань користувач може звернутися до служби підтримки, скориставшись інформацією, розміщеною в розділі «Контакти». Передбачено можливість надіслати повідомлення через вебформу або використати інші варіанти зв’язку (електронна пошта, телефон).

Ця інструкція є корисною, тому що підвищує зручність користування вебзастосунком, зменшує кількість помилок під час замовлення та загально покращує якість взаємодії з вебзастосунком. Такий підхід формує довіру до

онлайн-магазину, підвищує рівень задоволеності клієнтів і стимулює робити повторні покупки.

Висновки до третього розділу

В цьому розділі, було розроблено вебзастосунок для продажу продуктів харчування. Також була виконана фізична модель бази даних, що є головною складовою інтернет-магазину. Представлено основну сторінку, кошик, інформацію про вебзастосунок, продукти. Визнано, що користувацький інтерфейс – це візуальні та інтерактивні елементи цифрового продукту, такого як вебзастосунок або додаток, з якими користувачі взаємодіють безпосередньо. Зроблено проєктування користувацького інтерфейсу, архітектура вебзастосунку: у термінології розробки програмного забезпечення. Наступним кроком було створено сторінки вебзастосунку та його функціоналу: кожен має головну сторінку, яка називається Header (або шапка сайту) – це верхній блок на вебзастосунку. Шапка – це верхній елемент інтерфейсу, який відображається на всіх сторінках сайту. Потім адаптовано вебзастосунок під моделі iPhone SE та iPhone 12 Pro до Pixel 7, Samsung Galaxy S8+ і S20 Ultra, різних моделей iPad Mini, iPad Air, iPad Pro, Surface Pro, гнучких смартфонів, Galaxy Z Fold 5 і Asus ZenBook Fold, а також відносно бюджетних пристроїв Samsung A51/71 та розумних дисплеїв Nest Hub і Nest Hub Max. Планшети займають проміжне місце між телефонами, ноутбуками. Тому адаптація під планшети враховує значення як більший екран, ніж у телефонів, але менший і з іншою взаємодією, ніж у ноутбуків. Вирішено проблеми великих даних, що у вебзастосунку для продажу продуктів харчування, питання ефективної обробки великих даних, які є важливим фактором забезпечення стабільної роботи системи та зручності для користувача. Реалізовані механізми обробки даних дозволяють підтримувати ефективність системи при зростанні кількості товарів. Для цього реалізовано фільтрацію товарів за ключовими параметрами: ціною, категорією та рейтингом. Такий підхід забезпечує оптимальну роботу з великим обсягом інформації, забезпечуючи користувачеві швидкий доступ до найбільш

релевантних товарів. Спроєктованно БД, а саме що дані зберігаються у форматі документів або колекцій. MongoDB не використовує схеми, як це роблять реляційні БД, що покращує загальну продуктивність системи. Було протестовано роботу онлайн-магазину та його з’єднання до БД. Методи тестування охоплюють процес пошуку помилок або дефектів, а також перевірку програмних компонентів з метою їх оцінки. Та останнє в цьому розділі це інструкція користувача вебзастосунку, в якій наведено, як користувачеві здійснити повноцінне замовлення товарів та отримати необхідну інформацію.

ВИСНОВКИ

Під час виконання роботи: “Вебзастосунок для продажу продуктів харчування”, було застосовано основні підходи до проєктування інтернет-магазину, проаналізовано магазини-аналоги, визначено специфікацію та вимоги щодо дизайну, а також на основі сучасних технологій розроблено прототип інтернет-магазину. У роботі було використано такі методи дослідження, як аналіз наукової літератури та статистичних показників, порівняльний аналіз програмних рішень, моделювання структури вебзастосунку, а також практичне тестування функціоналу системи.

У першому розділі було описано предметну сферу, також оглянуто застосунки-аналоги, зазначено властивості та функції проєктної частини: використання онлайн-магазину є зручний онлайн доступ до асортименту товарів. Це дає користувачеві швидко обирати та замовляти потрібні продукти харчування.

У другому розділі обрано технології за якими створювався онлайн-магазин, розглянуто переваги та недоліки. Описувались методи та структура вебзастосунку, побудовано схеми. З'ясовано, що при розробці серверної частини вебзастосунку використовуються платформа Node.js, фреймворк Express.js і мова програмування JavaScript. Показано алгоритм пошуку та фільтрація товарів, та які кроки треба для цього. Виконано структуроване проєктування автоматизованої системи, що охоплює вибір оптимальних технологій реалізації, побудову логічно впорядкованої структури бази даних, визначення категорій і підкатегорій продуктів, а також формування алгоритму функціонування системи. Такий підхід забезпечує цілісність архітектури вебзастосунку, сприяє ефективній обробці інформації та підвищує зручність користування сервісом.

У третьому розділі було розроблено вебзастосунок для продажу продуктів харчування. Також була виконана фізична модель бази даних. Представлено основну сторінку, кошик, інформацію про вебзастосунок, продукти, зворотній зв'язок, команду, новини. Зроблено проєктування користувацького інтерфейсу,

наступним кроком було створено сторінки вебзастосунку та його функціоналу. Потім адаптовано вебзастосунок під різні моделі, та реалізовано механізми обробки даних, який дозволяє підтримувати ефективність системи при зростанні кількості товарів. Вирішено проблему великих даних, для цього реалізовано фільтрацію товарів за ключовими параметрами: ціною, категорією та рейтингом. Спроєктовано БД, після чого протестовано роботу онлайн-магазину та його з’єднання до БД. Наведено інструкцію користувача вебзастосунку, в якій наведено, як користувачеві здійснити повноцінне замовлення товарів та отримати необхідну інформацію.

Загалом, під час виконання дипломної роботи було застосовані вміння використовувати новітні засоби та методи для розробки онлайн-магазину. Реалізований вебзастосунок для продажу продуктів харчування відповідає всім встановленим вимогам та потребам, серед яких: резервування, редагування, видаляння даних, обробляти в БД, фільтрувати та формувати замовлення. Результати виконання роботи показує ґрунтовні знання з предметної сфери та вміння використовувати набуті навички для розробки високопродуктивних вебзастосунків у майбутньому.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Девід Фленган JavaScript 6 книга: Вид. 2-ге, 2017 рік, 1080 с.

2. Монтейро М. Дизайн – це робота: навчальна література/пер. з англ. Дарина Кирієнко. Іванов та Фербер, 2013. 176 с.

3. Understanding TF-IDF (Term Frequency-Inverse Document Frequency) URL: Understanding TF-IDF (Term Frequency-Inverse Document Frequency) | GeeksforGeeks (date of access 30.04.2025).

4. Andrew R., Keith J. HTML5 and CSS3: Modern for Web Development. San Francisco: A List Apart, 2017. 320p (date of access 20.04.2025).

5. JavaScript that scales [Електронний ресурс]: Режим доступу: https://www.typescriptlang.org/ (дата звернення: 10.04.2025).

6. Miller B. D. Principles of web design. Allworth, 2022. 280 p. (date of access 01.20.2025).

7. Hong P. Practical web design: Learn the fundamentals of web design with HTML5, CSS3, Bootstrap, JQuery, and Vue.js. Packt Publ., 2018. 368 p. (date of access 02.02.2025).

8. Mohamed Aladin Software Architecture – The Difference Between Architecture and Design. 27.07.2018. [Електронний ресурс] : Режим доступу: https://codeburst.io/software-architecture-the-difference-between-architecture-anddesign-7936abdd5830 (date of access 10.03.2025).

9. Azat Mardan Master Express.js: The Node.js Framework For Your Web Development. Apress, 2014. 372 с.

10. Куліковська К. С. Інформаційна система компонування, пошуку та генерації колірних схем: підручник. Миколаїв, Університетська книга, 2020 рік 16 с. https://krs.chmnu.edu.ua/jspui/bitstream/123456789/1412/1/401%20Куліковська%20К.С.%20Автореферат.docx.pdf (дата звернення: 27.04.2025).

11. Автор невідомий, JavaScript Tutorial: 2018 рік 50 с.

https://www.tutorialspoint.com/javascript/javascript_tutorial.pdf (дата звернення: 28.04.2025).

12. Автор невідомий, MongoDB Tutorial: 2018 рік 23 с. https://www.tutorialspoint.com/mongodb/mongodb_tutorial.pdf (дата звернення: 29.04.2025).

13. Автор невідомий, Node.js Tutorial: 2020 рік, 145 с.

14. Java Script URL: https://uk.javascript.info (дата звернення: 01.05.2025).

15. Visual Studio Code URL: https://learn.microsoft.com/ru-ru/visualstudio/intellicode/intellicode-visual-studio-code (дата звернення: 02.05.2025).

16. CHMNU Вебсервіс офіційний сайт URL: https://chmnu.edu.ua (дата звернення: 27.04.2025).

17. GitHub *Вебсервіс для хостингу GitHub.* URL: https://github.com/ (дата звернення: 28.04.2025).

18. Кедлек Т. Адаптивний дизайн. Робимо сайти для будь-яких пристроїв: навчальна література. Пітер, 2013. 288 с.

19. Wal-Mart Stores, Inc. - Company Profile, Information, Business Description, History, Background Information on Wal-Mart Stores, Inc. URL: https://www.referenceforbusiness.com/history2/20/Wal-Mart-Stores-Inc.html (date of access 10.04.2025).

20. Target - Company Profile, Information, Business Description, History. URL: https://corporate.target.com/about/purpose-history/history-timeline (date of access 10.04.2025).

21. Processing a large dataset in less than 100 lines of Node.js with async.queue. URL: https://medium.com/the-node-js-collection/processing-a-large-dataset-in-less-than-100-lines-of-node-js-with-async-queue-9766a78fa088 (date of access 18.05.2025).

22. Dataset. URL: https://www.databricks.com/glossary/what-is-dataset (date of access 23.05.2025).

23. Grocery Inventory and Sales Dataset
URL: https://www.kaggle.com/datasets/salahuddinahmedshuvo/grocery-inventory-and-sales-dataset (date of access 19.05.2025).

24. Sam`s Club, Wal-Mart Stores, Inc. - Company Profile, Information, Business Description, History, Background Information. URL: https://corporate.walmart.com/about/samsclub/sams-club-history (date of access 11.04.2025).

25. Amazon Fresh - Wikipedia: вебсайт URL: https://en.wikipedia.org/wiki/Amazon_Fresh (дата звернення 15.04.2025).

26. Figma URL: https://www.figma.com (дата звернення: 26.04.2025).

27. The Progressive JavaScript Framework: вебсайт. URL: https://vuejs.org/ (дата звернення: 24.01.2025).

28. Вступ до Mongoose. URL:
https://code.tutsplus.com/uk/articles/anintroduction-to-mongoose-for-mongodb-and-nodejs--cms-29527 (дата звернення: 15.04.2025).

29. Фреймворк Express js. URL: https://expressjs.com (дата звернення: 10.03.2025).

30. How to implement Transactions in Mongoose & Node.Js (Express). URL: https://www.ultimateakash.com/blog-details/IiwzQGAKYAo=/How-toimplement-Transactions-in-Mongoose-&-Node.Js-(Express) (date of access 10.03.2025).

ДОДАТОК А

Лістинг програмного коду

index.js

```
import express from 'express';
import mongoose from 'mongoose';
import session from 'express-session';
import { create as exphbs } from 'express-handlebars';
import router from './routes/router.js';
import path from 'path';
import { fileURLToPath } from 'url';

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);
const PORT = process.env.PORT || 3000;
const app = express();
const hbs = exphbs({
  defaultLayout: 'main',
  extname: 'hbs'
});
hbs.handlebars.registerHelper('add', function (value, addition) {
  return value + addition;
});
hbs.handlebars.registerHelper('times', function (n, block) {
  let accum = '';
  for (let i = 0; i < n; ++i) {
    accum += block.fn(i);
  }
  return accum;
});
app.engine('hbs', hbs.engine);
app.set('view engine', 'hbs');
app.set('views', 'views');
app.use(
  session({
    secret: 'amar',
    saveUninitialized: true,
    resave: true
  })
);
app.use(express.urlencoded({ extended: true }));
app.use(express.static(path.join(__dirname, 'public')));
app.use(router);
async function start() {
  try {
    await mongoose.connect(
      // eslint-disable-next-line max-len
       'mongodb+srv://OlenaTrofymenko:sZHzh9r2W9Vg7xV@cluster0.r28zfpj.mongodb.net/
products',
      {
        useNewUrlParser: true
      }
    );

    app.listen(PORT, () => {
      console.log('Server has been started...');
    });
  } catch (e) {
    console.log(e);
```

```
  }
}
start();
```

router.js

```
import { Router } from 'express';
import qs from 'fast-querystring';
import DatabaseService from '../services/DatabaseService.js';
import DataParser from '../helpers/DataParser.js';
import getUserId from '../helpers/getUserId.js';

const router = Router();
router.get('/', async (req, res) => {
  const userId = getUserId(req);
  const cartPrice = await DatabaseService.carts.getCartPrice(userId);
  let categoriesProducts = await DatabaseService.products.getProducts(8);
  let offersProducts = await DatabaseService.products.getProducts(12);
  let feedbacks = await DatabaseService.feedbacks.getFeedbacks();
  let blogArticles = await DatabaseService.blogArticles.getBlogArticles(2);
  offersProducts = offersProducts.slice(8, 12);
  categoriesProducts = categoriesProducts.map((product) =>
    DataParser.prepareProduct(product)
  );

  offersProducts = offersProducts.map((product) =>
    DataParser.prepareProduct(product)
  );

  feedbacks = feedbacks.map((feedback) => DataParser.prepareFeedback(feedback));
  blogArticles = blogArticles.map((article) =>
    DataParser.prepareBlogArticle(article)
  );

  res.render('index', {
    title: 'Organic',
    mainStyles: '/css/index.min.css',
    additionalStyles: [
      'https://cdnjs.cloudflare.com/ajax/libs/tiny-slider/2.9.4/tiny-slider.css'
    ],
    scriptApp: '/js/indexApp.js',
    categoriesProducts,
    offersProducts,
    feedbacks,
    blogArticles,
    cartPrice
  });
});

router.post('/search', async (req, res) => {
  const keywords = req.body.keywords;
  const query = qs.stringify({
    'product-name': keywords
  });
  res.redirect(`/shop?${query}`);
})
router.post('/search', async (req, res) => {
  let keywords = req.body.keywords;
  keywords = keywords.split(' ').join('-');
  res.redirect(`/search/${keywords}`);
});

// router.get('/search', async (req, res) => {
```

```
// res.redirect('/shop');
//  return;
//});

// router.get('/search/:keywords', async (req, res) => {
//  const userId = getUserId(req);
//  const cartPrice = await DatabaseService.carts.getCartPrice(userId);
//  const keywords = req.params.keywords.split('-');
//  let foundProducts = await DatabaseService.products.searchProducts(keywords);

//  foundProducts = foundProducts.map((product) =>
//    DataParser.prepareProduct(product)
//  );
//  res.render('search-results', {
//    title: `Search: ${keywords}`,
//    mainStyles: '/css/shop.min.css',
//    scriptApp: '/js/shopApp.js',
//    foundProducts,
//    cartPrice
//  });
//});

router.get('/cart', async (req, res) => {
  const userId = getUserId(req);
  const cartPrice = await DatabaseService.carts.getCartPrice(userId);
  const cart = await DatabaseService.carts.getCart(userId);
  const products = [];
  for (const productId in cart.filling) {
    const product = await DatabaseService.products.getProduct(productId);
    products.push({
      ...DataParser.prepareProduct(product),
      count: cart.filling[productId]
    });

  }
  res.render('cart', {
    title: 'Cart',
    mainStyles: '/css/cart.min.css',
    scriptApp: '/js/cartApp.js',
    cartPrice,
    products,
    orderPlaced: req.query.orderPlaced
  });
});

router.post('/cart', async (req, res) => {
  const userId = getUserId(req);
  const {
    'first-name': firstName,
    'last-name': lastName,
    country,
    street,
    city,
    'postal-code': postalCode,
    phone,
    email,
    'payment-method': paymentMethod
  } = req.body;
  const userInfo = {
    firstName,
    lastName,
```

```
    country,
    street,
    city,
    postalCode,
    phone,
    email,
    paymentMethod
  };

  await DatabaseService.orders.placeOrder(userId, userInfo);
  res.redirect('/cart?orderPlaced=true');
});

router.get('/cart/:id', async (req, res) => {
  const userId = getUserId(req);
  const cartPrice = await DatabaseService.carts.getCartPrice(userId);
  const cart = await DatabaseService.carts.getCart(userId);
  let product = await DatabaseService.products.getProduct(req.params.id);
  if (product === null) {
    res.redirect('/404');
    return;
  }
  product = DataParser.prepareProduct(product);
  product = {
    ...product,
    count: cart.filling[product._id]
  };

  res.render('edit-product', {
    title: 'Cart',
    mainStyles: '/css/shop-single.min.css',
    scriptApp: '/js/shopSingleApp.js',
    cartPrice,
    product
  });
});

router.post('/cart/:id', async (req, res) => {
  const userId = getUserId(req);
  await DatabaseService.carts.changeProductQuantity(
    userId,
    req.params.id,
    Number.parseInt(req.body.quantity)
  );
  res.redirect('/cart');
});

router.get('/about', async (req, res) => {
  const userId = getUserId(req);
  const cartPrice = await DatabaseService.carts.getCartPrice(userId);
  let teammates = await DatabaseService.teammates.getTeammates(3);
  teammates = teammates.map((teammate) => DataParser.prepareTeammate(teammate));
  res.render('about', {
    title: 'About Us',
    mainStyles: '/css/about.min.css',
    scriptApp: '/js/aboutApp.js',
    teammates,
    cartPrice
  });
});
```

```
router.get('/shop', async (req, res) => {
  const userId = getUserId(req);
  const cartPrice = await DatabaseService.carts.getCartPrice(userId);
  const {
    'product-name': productName,
    category,
    rating,
    'price-min': priceMin,
    'price-max': priceMax
  } = req.query;
  let products = [];
  console.log(productName, category, rating, priceMin, priceMax);
  const filterQuery = { productName, category, rating, priceMin, priceMax };
  if (productName || category || rating || priceMin || priceMax) {
    products = await DatabaseService.products.filterProducts(filterQuery);
  } else {
    products = await DatabaseService.products.getProducts();
  }
  products = products.map((product) => DataParser.prepareProduct(product));
  console.log(products);
  // products = await DatabaseService.products.getProducts();
  // products = products.map((product) => DataParser.prepareProduct(product));
  res.render('shop', {
    title: 'Shop',
    mainStyles: '/css/shop.min.css',
    scriptApp: '/js/shopApp.js',
    products,
    cartPrice,
    filterQuery
  });
});

router.get('/shop/:id', async (req, res) => {
  const userId = getUserId(req);
  const cartPrice = await DatabaseService.carts.getCartPrice(userId);
  let product = await DatabaseService.products.getProduct(req.params.id);
  if (product === null) {
    res.redirect('/404');
    return;
  }
  product = DataParser.prepareProduct(product);
  let relatedProducts = await DatabaseService.products.getRelatedProducts(
    product,
    4
  );
  relatedProducts = relatedProducts.map((product) =>
    DataParser.prepareProduct(product)
  );

  res.render('product', {
    title: product.title,
    mainStyles: '/css/shop-single.min.css',
    scriptApp: '/js/shopSingleApp.js',
    product,
    relatedProducts,
    cartPrice,
    productAdded: req.query.productAdded
  });
});

router.post('/shop/:id', async (req, res) => {
```

```
  const userId = getUserId(req);
  const productId = req.params.id;
  const quantity = Number.parseInt(req.body.quantity);
  await DatabaseService.carts.addToCart(userId, productId, quantity);
  res.redirect(`/shop/${productId}?productAdded=true`);
});

router.get('/service', async (req, res) => {
  const userId = getUserId(req);
  const cartPrice = await DatabaseService.carts.getCartPrice(userId);
  res.render('service', {
    title: 'Services',
    mainStyles: '/css/service.min.css',
    scriptApp: '/js/serviceApp.js',
    cartPrice
  });
});

router.get('/service/detailed', async (req, res) => {
  const userId = getUserId(req);
  const cartPrice = await DatabaseService.carts.getCartPrice(userId);
  res.render('service-detailed', {
    title: 'Quality Standard',
    mainStyles: '/css/service-single.min.css',
    scriptApp: '/js/serviceSingleApp.js',
    cartPrice
  });
});

router.get('/projects', async (req, res) => {
  const userId = getUserId(req);
  const cartPrice = await DatabaseService.carts.getCartPrice(userId);
  let farms = await DatabaseService.farms.getFarms();
  farms = farms.map((farm) => DataParser.prepareFarm(farm));
  res.render('projects', {
    title: 'Projects',
    mainStyles: '/css/portfolio.min.css',
    scriptApp: '/js/portfolioApp.js',
    farms,
    cartPrice
  });
});

router.get('/projects/:id', async (req, res) => {
  const userId = getUserId(req);
  const cartPrice = await DatabaseService.carts.getCartPrice(userId);

  let farm = await DatabaseService.farms.getFarm(req.params.id);
  if (farm === null) {
    res.redirect('/404');
    return;
  }
  farm = DataParser.prepareFarm(farm, true);
  res.render('farm', {
    title: farm.name,
    mainStyles: '/css/portfolio-single.min.css',
    scriptApp: '/js/portfolioSingleApp.js',
    farm,
    cartPrice
  });
});
```

```
router.get('/team', async (req, res) => {
  const userId = getUserId(req);
  const cartPrice = await DatabaseService.carts.getCartPrice(userId);
  let teammates = await DatabaseService.teammates.getTeammates();
  teammates = teammates.map((teammate) => DataParser.prepareTeammate(teammate));
  res.render('team', {
    title: 'Our Team',
    mainStyles: '/css/team.min.css',
    scriptApp: '/js/teamApp.js',
    teammates,
    cartPrice
  });
});

router.get('/blog', async (req, res) => {
  const userId = getUserId(req);
  const cartPrice = await DatabaseService.carts.getCartPrice(userId);
  let articles = await DatabaseService.blogArticles.getBlogArticles(6);
  articles = articles.map((article) => DataParser.prepareBlogArticle(article));
  res.render('blog', {
    title: 'Recent News',
    mainStyles: '/css/blog.min.css',
    scriptApp: '/js/blogApp.js',
    articles,
    cartPrice
  });
});
router.get('/blog/:id', async (req, res) => {
  const userId = getUserId(req);
  const cartPrice = await DatabaseService.carts.getCartPrice(userId);
  let article = await DatabaseService.blogArticles.getBlogArticle(
    req.params.id
  );
  if (article === null) {
    res.redirect('/404');
    return;
  }
  article = DataParser.prepareBlogArticle(article, true);
  res.render('article', {
    title: article.title,
    mainStyles: '/css/blog-single.min.css',
    scriptApp: '/js/blogSingleApp.js',
    article,
    cartPrice
  });
});
router.get('/contact', async (req, res) => {
  const userId = getUserId(req);
  const cartPrice = await DatabaseService.carts.getCartPrice(userId);
  res.render('contact', {
    title: 'Contact Us',
    mainStyles: '/css/contact.min.css',
    scriptApp: '/js/contactApp.js',
    cartPrice,
    messageSent: req.query.messageSent
  });
});
router.post('/contact', async (req, res) => {
  await DatabaseService.messages.sendMessage(req);
  res.redirect('/contact?messageSent=true');
```

```
});
router.post('/subscribe', async (req, res) => {
  const email = req.body.email;
  await DatabaseService.subscribers.addSubscriber(email);
  res.redirect(`back`);
});
router.get('*', async (req, res) => {
  const userId = getUserId(req);
  const cartPrice = await DatabaseService.carts.getCartPrice(userId);
  res.status(404).render('404', {
    title: '404 Not Found',
    mainStyles: '/css/404.min.css',
    scriptApp: '/js/404App.js',
    cartPrice
  });
});
router.get('/404', async (req, res) => {
  const userId = getUserId(req);
  const cartPrice = await DatabaseService.carts.getCartPrice(userId);

  res.status(404).render('404', {
    title: '404 Not Found',
    mainStyles: '/css/404.min.css',
    scriptApp: '/js/404App.js',
    cartPrice
  });
});
export default router;
```

Product.js

```
import { Schema, model } from 'mongoose';
const schema = new Schema({
  title: {
    type: String,
    required: true
  },
  category: {
    type: String,
    required: true
  },
  oldPrice: {
    type: Number,
    required: true
  },
  price: {
    type: Number,
    required: true
  },
  rating: {
    type: Number,
    required: true
  },
  imgName: {
    type: String,
    required: true
  },
  promoInfo: {
    type: String,
    required: true
  },
  description: {
    type: String,
```

```
    required: true
  },
  additionalInfo: {
    type: String,
    required: true
  }
});
export default model('Product', schema);
```

Farm.js

```
import { Schema, model } from 'mongoose';
const schema = new Schema({
  name: {
    type: String,
    required: true
  },
  type: {
    type: String,
    required: true
  },
  promoImageNameMini: {
    type: String,
    required: true
  },
  promoImageName: {
    type: String,
    required: true
  },
  description: {
    type: String,
    required: true
  },
  date: {
    type: Date,
    required: true
  },
  clientName: {
    type: String,
    required: true
  },
  category: {
    type: String,
    required: true
  },
  service: {
    type: String,
    required: true
  },
  content: {
    type: String,
    required: true
  }
});
export default model('Farm', schema);
```

DatabaseService.js

```
import Product from './models/Product.js';
import Feedback from './models/Feedback.js';
import BlogArticle from './models/BlogArticle.js';
import Farm from './models/Farm.js';
import Teammate from './models/Teammate.js';
import Message from './models/Message.js';
```

```
import Cart from './models/Cart.js';
import Order from './models/Order.js';
import Subscriber from './models/Subscriber.js';
const DatabaseService = {
  products: {
    getProducts: async (limit = null) => {
      return await Product.find({}).limit(limit).lean();
    },
    getProduct: async (id) => {
      let product;
      try {
        product = await Product.findOne({ _id: id }).lean();
        if (product === null) {
          throw ReferenceError("Product with such id doesn't exist!");
        }
      } catch (err) {
        console.log(err);
        return null;
      }
      return product;
    },
    getRelatedProducts: async (product, limit = null) => {
      return await Product.find({
        category: product.category,
        _id: { $ne: product._id }
      })
        .limit(limit)
        .lean();
    },
    filterProducts: async (filterQuery) => {
      const categoryMap = {
        'grains-pulses': 'Grains & Pulses',
        beverages: 'Beverages',
        'fruits-vegetables': 'Fruits & Vegetables',
        'oils-fats': 'Oils & Fats',
        dairy: 'Dairy',
        bakery: 'Bakery',
        seafood: 'Seafood'
      };

      let filteredProducts = [];
      const { productName, category, rating, priceMin, priceMax } = filterQuery;
      const filter = {};
      if (productName) {
        filter.title = { $regex: productName, $options: 'i' };
      }
      if (category) {
        filter.category = categoryMap[category];
      }
      if (rating) {
        filter.rating = { $gt: rating - 1 };
      }
      if (priceMin || priceMax) filter.price = {};
      if (priceMin) {
        filter.price.$gte = priceMin;
      }
      if (priceMax) {
        filter.price.$lte = priceMax;
      }
      filteredProducts = await Product.find(filter);
      filteredProducts = filteredProducts.map((product) => {
```

```
      return { ...product._doc };
    });
    return filteredProducts;
  },

  searchProducts: async (keywords) => {
    let foundProducts = [];
    for (const keyword of keywords) {
      const foundProductsByName = await Product.find({
        title: { $regex: keyword, $options: 'i' }
      }).lean();
      foundProducts = foundProducts.concat(foundProductsByName);
      const foundProductsByCategory = await Product.find({
        category: { $regex: keyword, $options: 'i' }
      }).lean();
      foundProducts = foundProducts.concat(foundProductsByCategory);
    }
    foundProducts = foundProducts.filter((product, index, self) => {
      return (
        index ===
        self.findIndex((prod) => {
          return prod._id.toString() === product._id.toString();
        })
      );
    });

    return foundProducts;
  }
},

feedbacks: {
  getFeedbacks: async (limit = 3) => {
    return await Feedback.find({}).limit(limit).lean();
  }
},
blogArticles: {
  getBlogArticles: async (limit = null) => {
    return await BlogArticle.find({}).limit(limit).lean();
  },

  getBlogArticle: async (id) => {
    let article;
    try {
      article = await BlogArticle.findOne({ _id: id }).lean();
      if (article === null) {
        throw ReferenceError("Article with such id doesn't exist!");
      }
    } catch (err) {
      console.log(err);
      return null;
    }
    return article;
  }
},

farms: {
  getFarms: async () => {
    return await Farm.find({}).lean();
  },
  getFarm: async (id) => {
    let farm;
```

```
    try {
      farm = await Farm.findOne({ _id: id }).lean();
      if (farm === null) {
        throw ReferenceError("Farm with such id doesn't exist!");
      }
    } catch (err) {
      console.log(err);
      return null;
    }
    return farm;
  }
},

teammates: {
  getTeammates: async (limit = null) => {
    return await Teammate.find({}).limit(limit).lean();
  }
},

messages: {
  sendMessage: async (req) => {
    const message = new Message({
      fullName: req.body.name,
      email: req.body.email,
      company: req.body.company,
      subject: req.body.subject,
      message: req.body.message
    });
    await message.save();
  }
},
carts: {
  getCart: async (userId) => {
    const doesCartExist = await Cart.exists({ userId });
    if (!doesCartExist) {
      const cart = new Cart({
        userId,
        filling: {}
      });
      await cart.save();
    }
    return await Cart.findOne({ userId }).lean();
  },

  addToCart: async (userId, productId, quantity) => {
    if (quantity === 0) {
      return;
    }
    const cart = await Cart.findOne({ userId }).lean();
    const filling = cart.filling ? cart.filling : {};
    filling[productId] = filling[productId]
      ? filling[productId] + quantity
      : quantity;
    await Cart.updateOne({ userId }, { filling });
  },
  getCartPrice: async (userId) => {
    let cart = await Cart.findOne({ userId }).lean();
    if (!cart) {
      cart = await DatabaseService.carts.getCart(userId);
    }
    let price = 0;
```

```
      if (cart.filling) {
        for (const productId in cart.filling) {
          price +=
            (await Product.findOne({ _id: productId }).lean()).price *
            cart.filling[productId];
        }
      }
      return price;
    },
    changeProductQuantity: async (userId, productId, quantity) => {
      const cart = await Cart.findOne({ userId }).lean();
      const filling = cart.filling;
      if (quantity !== 0) {
        filling[productId] = quantity;
      } else {
        delete filling[productId];
      }
      await Cart.updateOne({ userId }, { filling });
    }
  },
  orders: {
    placeOrder: async (userId, userInfo) => {
      const cart = await Cart.findOne({ userId }).lean();
      const order = new Order({
        userId,
        filling: cart.filling,
        price: await DatabaseService.carts.getCartPrice(userId),
        userInfo
      });
      await order.save();
      await Cart.updateOne({ userId }, { filling: {} });
    }
  },
  subscribers: {
    addSubscriber: async (email) => {
      const doesSubscriberExist = await Subscriber.exists({ email: email });
      if (!doesSubscriberExist) {
        const subscriber = new Subscriber({ email: email });
        await subscriber.save();
      }
    }
  }
};
export default DatabaseService;
```

Gulpfile.js

```
import fs from 'fs';
import path from 'path';
import os from 'os';
import gulp from 'gulp';
import browserSync from 'browser-sync';

import htmlmin from 'gulp-htmlmin';
import dartSass from 'sass';
import gulpSass from 'gulp-sass';
const sass = gulpSass(dartSass);
import rename from 'gulp-rename';
import autoprefixer from 'gulp-autoprefixer';
import GulpCleanCss from 'gulp-clean-css';
import webpackStream from 'webpack-stream';
import imagemin from 'gulp-imagemin';
```

```
import imageminPngquant from 'imagemin-pngquant';
import imageminZopfli from 'imagemin-zopfli';
import imageminSvgo from 'imagemin-svgo';
import imageminMozjpeg from 'imagemin-mozjpeg';
import imageminGiflossy from 'imagemin-giflossy';

import plumber from 'gulp-plumber';
import notify from 'gulp-notify';
import fonter from 'gulp-fonter';
import ttf2woff from 'gulp-ttf2woff';
import ttf2woff2 from 'gulp-ttf2woff2';
import newer from 'gulp-newer';
import webp from 'gulp-webp';
import fileInclude from 'gulp-file-include';
import webpHtmlNosvg from 'gulp-webp-html-nosvg';
import replace from 'gulp-replace';
import groupCssMediaQueries from 'gulp-group-css-media-queries';
import webpcss from 'gulp-webpcss';

const distPath = path.resolve('./dist');
const srcPath = path.resolve('./src');

const distHtmlPath = distPath;
const srcHtmlPath = srcPath;
const cssDir = 'css';
// const cssPreprocessorType = 'sass'
const cssPreprocessorType = 'scss';
const distStylesPath = path.join(distPath, cssDir);
const srcStylesPath = path.join(srcPath, cssPreprocessorType);

const jsDir = 'js';
const distScriptsPath = path.join(distPath, jsDir);
const srcScriptsPath = path.join(srcPath, jsDir);
const scriptsToBundle = {
  indexApp: path.join(srcScriptsPath, 'index.js'),
  aboutApp: path.join(srcScriptsPath, 'about.js'),
  shopApp: path.join(srcScriptsPath, 'shop.js'),
  cartApp: path.join(srcScriptsPath, 'cart.js'),
  shopSingleApp: path.join(srcScriptsPath, 'shop-single.js'),
  serviceApp: path.join(srcScriptsPath, 'service.js'),
  serviceSingleApp: path.join(srcScriptsPath, 'service-single.js'),
  portfolioApp: path.join(srcScriptsPath, 'portfolio.js'),
  portfolioSingleApp: path.join(srcScriptsPath, 'portfolio-single.js'),
  teamApp: path.join(srcScriptsPath, 'team.js'),
  blogApp: path.join(srcScriptsPath, 'blog.js'),
  blogSingleApp: path.join(srcScriptsPath, 'blog-single.js'),
  contactApp: path.join(srcScriptsPath, 'contact.js'),
  '404App': path.join(srcScriptsPath, '404.js')
};

const fontsDir = 'fonts';
const distFontsPath = path.join(distPath, fontsDir);
const srcFontsPath = path.join(srcPath, fontsDir);
const fontsFile = `${srcStylesPath}/base/_fonts.scss`;

const iconsDir = 'icons';
const distIconsPath = path.join(distPath, iconsDir);
const srcIconsPath = path.join(srcPath, iconsDir);

const imgDir = 'img';
const distImagesPath = path.join(distPath, imgDir);
```

```
const srcImagesPath = path.join(srcPath, imgDir);
const bsPort = 3000;

gulp.task('html', processHtml);
gulp.task('styles', processStyles);
gulp.task('scripts', processScriptsDev);
gulp.task('otfToTtf', processFontsOtfToTtf);
gulp.task('ttfToWoff', processFontsTtfToWoff);
gulp.task('fontsStyle', processFontsStyle);
gulp.task('fonts', gulp.series('otfToTtf', 'ttfToWoff', 'fontsStyle'));
gulp.task('icons', () => processPictures(srcIconsPath, distIconsPath, 'ICONS'));
gulp.task('images', () =>
  processPictures(srcImagesPath, distImagesPath, 'IMAGES')
);
gulp.task(
  'dist-sync',
  gulp.parallel('html', 'styles', 'scripts', 'fonts', 'icons', 'images')
);
gulp.task('watch', watch);
gulp.task('build-prod-scripts', processScriptsProd);
gulp.task('default', gulp.parallel('watch'));

function fixPathForGulpWatch(oldPath) {
  const pathString = oldPath;
  const fix = /\\{1,}|\/{1,}/;
  const userHome = os.homedir();

  return pathString
    .replace(new RegExp(fix, 'gm'), '/')
    .replace(new RegExp(fix, 'gm'), '/')
    .replace('~', userHome);
}
function processHtml() {
  return gulp
    .src(`${srcHtmlPath}/*.html`)
    .pipe(
      plumber(
        notify.onError({
          title: 'HTML',
          message: 'Error: <%= error.message %>'
        })
      )
    )
    .pipe(fileInclude())
    .pipe(replace(/@img\//g, 'img/'))
    .pipe(replace(/@icons\//g, 'icons/'))
    .pipe(webpHtmlNosvg())
    .pipe(htmlmin({ collapseWhitespace: true }))
    .pipe(gulp.dest(distHtmlPath))
    .pipe(browserSync.stream());
}
function processStyles() {
  return gulp
    .src(`${srcStylesPath}/**/*.+(scss|sass)`)
    .pipe(
      plumber(
        notify.onError({
          title: 'SCSS',
          message: 'Error: <%= error.message %>'
        })
      )
```

```
    )
    .pipe(
      sass({
        outputStyle: 'compressed'
      })
    )
    .pipe(groupCssMediaQueries())
    .pipe(
      webpcss({
        webpClass: '.webp',
        noWebpClass: '.no-webp'
      })
    )
    .pipe(
      autoprefixer({
        grid: true,
        overrideBrowserslist: ['last 3 versions'],
        cascade: true
      })
    )
    .pipe(GulpCleanCss())
    .pipe(
      rename({
        extname: '.min.css'
      })
    )
    .pipe(replace(/@img\//g, '../img/'))
    .pipe(replace(/@icons\//g, '../icons/'))
    .pipe(gulp.dest(distStylesPath))
    .pipe(browserSync.stream());
}
function processScriptsDev() {
  return gulp
    .src(`${srcScriptsPath}/**/*.js`)
    .pipe(
      webpackStream({
        mode: 'development',
        entry: scriptsToBundle,
        output: {
          filename: '[name].js',
          path: distScriptsPath
        },
        watch: false,
        devtool: 'source-map',
        module: {
          rules: [
            {
              test: /\.m?js$/,
              exclude: /(node_modules|bower_components)/,
              use: {
                loader: 'babel-loader',
                options: {
                  presets: [
                    [
                      '@babel/preset-env',
                      {
                        debug: true,
                        corejs: 3,
                        useBuiltIns: 'usage'
                      }
                    ]
```

```
                  ]
                }
              }
            }
          ]
        }
      })
    )
    .pipe(gulp.dest(distScriptsPath))
    .pipe(browserSync.stream());
}
function processFontsOtfToTtf() {
  if (!fs.existsSync(fontsFile)) {
    return gulp
      .src(`${srcFontsPath}/*.otf`, {})
      .pipe(
        plumber(
          notify.onError({
            title: 'FONTS',
            message: 'Error: <%= error.message %>'
          })
        )
      )
      .pipe(
        fonter({
          formats: ['ttf']
        })
      )
      .pipe(gulp.dest(`${srcFontsPath}`));
  } else {
    return gulp.src(`${srcPath}`);
  }
}
function processFontsTtfToWoff() {
  if (!fs.existsSync(fontsFile)) {
    return gulp
      .src(`${srcFontsPath}/*.ttf`)
      .pipe(
        plumber(
          notify.onError({
            title: 'FONTS',
            message: 'Error: <%= error.message %>'
          })
        )
      )
      .pipe(ttf2woff())
      .pipe(gulp.dest(`${distFontsPath}`))
      .pipe(gulp.src(`${srcFontsPath}/*.ttf`))
      .pipe(ttf2woff2())
      .pipe(gulp.dest(`${distFontsPath}`));
  } else {
    return gulp.src(`${srcPath}`);
  }
}
function processFontsStyle() {
  // eslint-disable-next-line no-empty-function
  function cb() {}
  fs.readdir(distFontsPath, (err, fontsFiles) => {
    if (fontsFiles) {
      if (!fs.existsSync(fontsFile)) {
        fs.writeFile(fontsFile, '', cb);
```

```
        let newFileOnly;
        for (let i = 0; i < fontsFiles.length; i++) {
          const fontFileName = fontsFiles[i].split('.')[0];
          if (newFileOnly !== fontFileName) {
            const fontName = fontFileName.split('-')[0]
              ? fontFileName.split('-')[0]
              : fontFileName;

            let fontWeight = fontFileName.split('-')[1]
              ? fontFileName.split('-')[1].toLowerCase()
              : fontFileName.toLowerCase();

            let fontStyle = 'normal';
            if (fontWeight.includes('italic')) {
              fontStyle = 'italic';
              fontWeight = fontWeight.substring(
                0,
                fontWeight.indexOf('italic')
              );
            }
            switch (fontWeight) {
              case 'thin':
                fontWeight = 100;
                break;
              case 'extralight':
                fontWeight = 200;
                break;
              case 'light':
                fontWeight = 300;
                break;
              case 'medium':
                fontWeight = 500;
                break;
              case 'semibold':
                fontWeight = 600;
                break;
              case 'bold':
                fontWeight = 700;
                break;
              case 'extrabold':
              case 'heavy':
                fontWeight = 800;
                break;
              case 'black':
                fontWeight = 900;
                break;
              default:
                fontWeight = 400;
            }
            fs.appendFile(
              fontsFile,
              // eslint-disable-next-line max-len
                    `@font-face  {\n   font-family:  ${fontName};\n   font-display:
swap;\n       src:      url("../fonts/${fontFileName}.woff2")      format("woff2"),
url("../fonts/${fontFileName}.woff")        format("woff");\n          font-weight:
${fontWeight};\n  font-style: ${fontStyle};\n}\r\n`,
              cb
            );
            newFileOnly = fontFileName;
          }
        }
```

```
      } else {
        console.log(
          // eslint-disable-next-line max-len
          'File scss/base/_fonts.scss already exists. Remove it if you want to update
your fonts.'
        );
      }
    }
  });
  return gulp.src(`${srcPath}`);
}
function processPictures(srcPicturesFolder, distPicturesFolder, errorTitle) {
  return gulp
    .src(`${srcPicturesFolder}/**/*.{jpg,jpeg,png,gif,webp,svg}`)
    .pipe(
      plumber(
        notify.onError({
          title: errorTitle,
          message: 'Error: <%= error.message %>'
        })
      )
    )
    .pipe(newer(distPicturesFolder))
    .pipe(webp())
    .pipe(gulp.dest(distPicturesFolder))
    .pipe(gulp.src(`${srcPicturesFolder}/**/*`))
    .pipe(newer(distPicturesFolder))
    .pipe(
      imagemin([
        imageminPngquant({
          speed: 1,
          quality: [0.95, 1]
        }),
        imageminZopfli({
          more: true
        }),
        imageminGiflossy({
          optimizationLevel: 3,
          optimize: 3,
          lossy: 2
        }),
        imageminSvgo({
          plugins: [{ removeViewBox: true }, { cleanupIDs: false }]
        }),
        imageminMozjpeg({
          quality: 90,
          progressive: true
        })
      ])
    )
    .pipe(gulp.dest(distPicturesFolder))
    .pipe(browserSync.stream());
}
function watch() {
  gulp.parallel('dist-sync')();
  browserSync.init({
    server: distPath,
    port: bsPort,
    ui: {
      port: bsPort + 1
    },
```

```
    notify: true
  });
  gulp
    .watch(`${fixPathForGulpWatch(srcHtmlPath)}/**/*.html`)
    .on('all', gulp.parallel('html'));
  gulp.watch(
    `${fixPathForGulpWatch(srcStylesPath)}/**/*.+(scss|sass|css)`,
    gulp.parallel('styles')
  );
  gulp
    .watch(`${fixPathForGulpWatch(srcScriptsPath)}/**/*.js`)
    .on('change', gulp.parallel('scripts'));
  gulp
    .watch(`${fixPathForGulpWatch(srcFontsPath)}/**/*`)
    .on('all', gulp.parallel('fonts'));
  gulp
    .watch(`${fixPathForGulpWatch(srcIconsPath)}/**/*`)
    .on('all', gulp.parallel('icons'));
  gulp
    .watch(`${fixPathForGulpWatch(srcImagesPath)}/**/*`)
    .on('all', gulp.parallel('images'));
}
function processScriptsProd() {
  return gulp
    .src(`${srcScriptsPath}/**/*.js`)
    .pipe(
      webpackStream({
        mode: 'production',
        entry: scriptsToBundle,
        output: {
          filename: '[name].js',
          path: distScriptsPath
        },
        module: {
          rules: [
            {
              test: /\.m?js$/,
              exclude: /(node_modules|bower_components)/,
              use: {
                loader: 'babel-loader',
                options: {
                  presets: [
                    [
                      '@babel/preset-env',
                      {
                        corejs: 3,
                        useBuiltIns: 'usage'
                      }
                    ]
                  ]
                }
              }
            }
          ]
        }
      })
    )
    .pipe(gulp.dest(distScriptsPath));
}
```