Полтавський університет економіки і торгівлі
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп’ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)
«___»_____202__р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

РОЗРОБКА ЕЛЕМЕНТІВ ІНТЕРФЕЙСУ ДЛЯ ПРОДУКТОВОГО ІНТЕРНЕТ-МАГАЗИНУ «Ecobazar»

зі спеціальності 122 «Комп’ютерні науки»
освітня програма «Комп’ютерні науки»
ступеня бакалавра

Виконавець роботи Сіваченко Роман Ігорович
_____«___»_____202__р.
(підпис)

Науковий керівник к.ф.-м.н., доц., Ольховська Олена Володимирівна
_____«___»_____202__р.
(підпис)

Резидент _____

ПОЛТАВА – 2025

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Ольховська О.В.
«_ » _____2025р.

ЗАВДАННЯ ТА КАЛЕНДАРНИЙ ГРАФІК ВИКОНАННЯ ДИПЛОМНОЇ РОБОТИ

на тему «Розробка елементів інтерфейсу для продуктового інтернет-магазину Ecobazar»
зі спеціальності 122 «Комп'ютерні науки»
освітня програма «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Сіваченко Роман Ігорович
Затверджена наказом ректора № ____-Н від «___» _____ 202_ р.
Термін подання студентом роботи «__ » _____ 202_ р.
Вихідні дані до кваліфікаційно роботи: публікації з теми, веб-сайти сайтів кафедр.
Зміст пояснювальної записки (перелік питань, які потрібно розробити)

СПИСОК ЛІТЕРАТУРИ
ДОДАТОК А. Код програми
ДОДАТОК Б. Перелік тестових питань
Перелік графічного матеріалу: 25 рисунків

Консультанти розділів кваліфікаційної роботи

Розділ	ПІБ, посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Постанова задачі	Ольховська О.В.		
Інформаційний огляд	Ольховська О.В.		
Теоретична частина	Ольховська О.В.		
Практична частина	Ольховська О.В.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
1. Вступ		
2. Вивчення методичних рекомендацій та стандартів та звіт керівнику		
3 .Постановка задачі		
4. Інформаційний огляд джерел бібліотек та інтернету		
5. Теоретична частина		
6 .Практична частина		
7. Закінчення оформлення		
8. Доповідь студента на кафедрі		
9. Доопрацювання (за необхідністю), рецезування		

Дата видачі завдання «____»_____ 202__ р.
Здобувач вищої освіти Сіваченко Роман Ігорович
Науковий керівник Ольховська О.В.

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на_____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 202_ р.
Секретар ЕК_____ _____
(підпис) (ініціал та прізвище)

Затверджую
Зав. кафедрою_____
к.ф.-м.н. Олена Ольховська
«____» _____ 202_ р.

Погоджено
Науковий керівник_____
к.ф.-м.н. Ольховська О.В.
«____» _____ 202_ р.

План

кваліфікаційної роботи на тему
«Розробка елементів інтерфейсу для продуктового інтернет-магазину Ecobazar»
зі спеціальності 122 Комп'ютерні науки
освітня програма 122 «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Сіваченко Роман Ігорович

ДОДАТОК Б. Перелік тестових питань

Здобувач вищої освіти _____ _____

(підпис)

«____» _____ 2025 р.

РЕФЕРАТ

Записка: 75 стр., 25 рис., 2 додатки, 14 джерел.

Об’єкт розробки – інтерфейс користувача для сучасного веб-додатку, реалізованого із використанням технологій React, TypeScript та SCSS.

Тема роботи - розробка адаптивного, функціонального та візуально привабливого веб-інтерфейсу для демонстрації товарів та взаємодії з користувачем.

Мета роботи – створення повноцінного фронтенд-застосунку з чистим та масштабованим кодом, що відповідає сучасним стандартам UI/UX дизайну та забезпечує зручну навігацію користувача

Методи дослідження та інформаційне забезпечення – середовище розробки Visual Studio Code, мова програмування TypeScript, бібліотека React, SCSS-модулі для стилізації, бібліотека Swiper для слайдерів, модульне структурування компонентів, використання Git для контролю версій. Додатково застосовано бібліотеки classNames, react-icons, інструменти роботи з графікою у форматі SVG, а також застосовано адаптивну верстку згідно з принципами Mobile First.

Предмет дослідження - процес побудови сучасного UI для веб-додатку, що демонструє товари та динамічний контент (банери, акційні пропозиції, рейтинг товарів, відгуки тощо), із використанням функціональних компонентів React та гнучкої стилізації через SCSS.

Результати дослідження – реалізовано повноцінний інтерфейс інтернет-магазину з поділом на компоненти, підтримкою адаптивності, керуванням даними через окремі файли та підтримкою повторного використання стилів. Створено модулі для відображення банерів, товарів різних типів (великі та малі карточки), рейтингу, відгуків клієнтів через **Swiper**-слайдер, а також навігаційні елементи з урахуванням

доступності. Забезпечено чисту структуру коду з дотриманням принципів компонентного підходу, розширюваності та повторного використання.

В результаті тестування встановлено, що веб-інтерфейс, маршрутизація, динамічні компоненти, анімації, інтеграція слайдерів, відображення контенту з масивів та управління станом за допомогою Redux — працюють стабільно та коректно на всіх етапах взаємодії користувача з сайтом**.**

Ключові слова: веб-інтерфейс, React, TypeScript, Redux, SCSS, модульна верстка, UI/UX, адаптивний дизайн, компоненти, Swiper, слайдер, фронтенд, інтернет-магазин, товари, рейтинг, банер, навігація, SVG, classNames, структура проєкту, код-сплітінг, респонсивність, WebPack, Git.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	**Пояснення умовних позначень, символів, скорочень**
HTML	Мова розмітки гіпертексту
CSS	Мова стилів для опису зовнішнього вигляду веб-сторінок
SCSS	Розширення CSS з підтримкою змінних, вкладеності та міксинів
JS	JavaScript — мова програмування для інтерактивності веб-сторінок
TS / TypeScript	Надбудова над JavaScript з підтримкою статичної типізації
React	JavaScript-бібліотека для створення інтерфейсів користувача
Redux	Бібліотека для керування станом додатку
JSX	Синтаксис, що дозволяє писати HTML-подібний код у JavaScript
UI / UX	Інтерфейс користувача / Досвід користувача
SPA	Single Page Application — односторінковий веб-додаток
API	Application Programming Interface — інтерфейс прикладного програмування
SVG	Векторний графічний формат для зображень
DOM	Document Object Model — об’єктна модель документа
IDE	Інтегроване середовище розробки
VS Code	Visual Studio Code — редактор коду від Microsoft
Git	Розподілена система керування версіями

Swiper	Бібліотека для реалізації адаптивних слайдерів
npm	Node Package Manager — система керування пакетами для JavaScript
Фреймворк	Готовий набір інструментів і бібліотек для спрощення розробки програмного забезпечення

ВСТУП

Актуальність. У сучасному світі інтернет-магазини стали невід'ємною частиною життя, особливо коли мова йде про продукти харчування. Завдяки таким платформам люди можуть економити час, обираючи товари онлайн та отримуючи їх у зручний спосіб. Ecobazar — це продуктова онлайн-крамниця, що пропонує екологічно чисту продукцію. Розробка frontend-частини магазину є важливим завданням, адже саме від зручності та дизайну залежить досвід користувачів і, відповідно, успіх проєкту. Ця тема особливо актуальна, оскільки попит на екологічні товари постійно зростає, а сучасні технології дозволяють створювати прості та функціональні рішення.

Об'єктом розробки є Інтернет-магазин Ecobazar, який спеціалізується на продажу продуктів харчування та товарів екологічного спрямування, доступних для широкого кола споживачів.

Предметом розробки є frontend-частина продуктового інтернет-магазину Ecobazar, яка включає створення користувацького інтерфейсу, та впровадження необхідного функціоналу, такого як каталог товарів, корзина покупок і форма реєстрації користувача.

Результатом проєкту є завершена frontend-частина інтернет-магазину Ecobazar, яка буде зручною у використанні, забезпечуватиме швидкий доступ до всіх необхідних функцій та привабливий дизайн. Це дозволить підвищити зручність для користувачів і сприятиме популяризації екологічно чистих продуктів.

ПОСТАНОВКА ЗАДАЧІ

Головною метою проєкту є створення вебсайту для продуктового інтернет-магазину Ecobazar. Цей ресурс має забезпечити користувачам зручний доступ до асортименту товарів, надання інформації про продукцію, а також можливість швидкого й простого оформлення замовлень.

Основні цілі дослідження та розробки полягають у створенні інтерфейсу, зрозумілого для користувачів, з акцентом на зручність і швидкий доступ до всіх функцій магазину. Також важливим є впровадження сучасних вебтехнологій, які забезпечать стабільну, надійну й продуктивну роботу додатку в реальних умовах.

Завдання проєкту включають:

- Побудову чіткої структури сайту, яка охоплює головну сторінку, каталог продукції, сторінки реєстрації, кошик, форму замовлення, а також реалізацію швидкого перегляду товару з галереєю фотографій у вигляді модального вікна.
- Вибір ефективних інструментів розробки:
    - React – для створення динамічних компонентів,
    - TypeScript – для покращення надійності коду,
    - Redux Toolkit – для централізованого управління станом (зокрема, кошиком),
    - Sass – для написання масштабованих і читабельних стилів,
    - Swiper – для реалізації інтерактивних слайдерів (наприклад, акцій або новинок).
- Реалізацію основного функціоналу:
    - сортування товарів за ціною, типом або новизною,
    - фільтрацію за категоріями, акціями або наявністю знижок,
    - можливість швидкого перегляду товару з додаванням у кошик,
    - форму реєстрації для створення акаунту перед оформленням покупки.

- Проведення тестування компонентів на відповідність функціональним вимогам і загальну зручність користування.

Очікуваний результат – розроблений сайт інтернет-магазину Ecobazar із широкими функціональними можливостями: перегляд каталогу, сортування та фільтрація товарів, швидкий перегляд, додавання до корзини та зручна реєстрація. Такий підхід сприятиме залученню клієнтів і розвитку онлайн-продажів.

РОЗДІЛ 1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Особливості та виклики створення інтерфейсу для онлайн-магазину

Розробка інтерфейсної частини веб-додатків є ключовим етапом у процесі створення сучасних онлайн-сервісів. це не лише про візуальну привабливість, а й про забезпечення функціональної, зручної та інтуїтивної взаємодії користувача із системою. Врахування принципів UX/UI-дизайну, технічних обмежень і поведінки споживачів є ключовими факторами у процесі розробки.

До основних викликів при створенні інтерфейсу інтернет-магазину можна віднести:

- **Раціональний вибір технологій**. Від обраного стеку залежить гнучкість проєкту, його масштабованість та швидкість реалізації. Використання надлишково складних або застарілих рішень може ускладнити розробку і підтримку.
- **Планування структури інтерфейсу**. Важливо заздалегідь продумати логіку розміщення основних елементів: каталогу, пошуку, фільтрів, кошика, особистого кабінету тощо. Без чіткої структури інтерфейс стає заплутаним для користувача.
- **Забезпечення адаптивності**. Сайт має коректно відображатись як на великих екранах, так і на мобільних пристроях. Адаптивна верстка вимагає ретельного тестування та правильного використання CSS-сіток, медіазапитів і компонентів.
- **Оптимізація продуктивності**. Надмірно об’ємні компоненти, необґрунтоване повторне рендерення, неефективне управління станом — усе це знижує швидкодію сайту. У випадку інтернет-магазину, де активно працюють фільтрація, пагінація, сортування та пошук, оптимізація критично важлива.
- **Інтеграція з бекендом**. Фронтенд повинен грамотно взаємодіяти з API, обробляти помилки, відображати завантаження даних та реагувати на зміну стану системи в реальному часі.

- **Підтримка та масштабування**. У проєкті, що розвивається, важливо забезпечити гнучкість структури компонентів і модулів, щоб легко додавати нові функції — наприклад, персоналізовані рекомендації або багатомовність.

Усе це потребує чіткого планування, вибору відповідного стеку технологій і врахування особливостей поведінки користувачів в електронній комерції. Саме тому створення навіть ззовні простого інтерфейсу вимагає глибокого аналізу, уважного дизайну та сучасних підходів до реалізації.

2.2 Технологічні засоби реалізації клієнтської частини інтернет-магазину

Сучасна фронтенд-розробка інтернет-магазинів базується на використанні високопродуктивних та гнучких інструментів. Для створення ефективного та підтримуваного інтерфейсу було обрано стек технологій, що відповідає вимогам як користувацького досвіду, так і внутрішньої організації коду.

До ключових інструментів, використаних у реалізації інтерфейсу, належать:

- **React** — JavaScript-бібліотека для побудови інтерфейсів користувача. Вона дозволяє компонувати UI із незалежних, повторно використовуваних елементів. React спрощує розробку завдяки декларативному стилю програмування та ефективному оновленню DOM через віртуальну модель (Virtual DOM).
- **TypeScript** — надмножина JavaScript, яка додає статичну типізацію. У проєктах середньої та високої складності використання TypeScript підвищує безпеку розробки, дозволяє уникати типових помилок та покращує підтримку коду в довготривалій перспективі.
- **Redux Toolkit** — сучасна офіційна бібліотека для керування станом у додатках React. Вона стандартизує підходи до роботи з глобальним станом, дозволяючи ефективно координувати дані між компонентами, зменшити надлишковість у коді та забезпечити передбачуваність у взаємодії з користувачем.

- **React Router** — бібліотека для маршрутизації, що дозволяє реалізувати багатосторінкову навігацію без перезавантаження сторінки. Це важливо для збереження інтерактивності та безперервності взаємодії з користувачем.
- **SCSS (Sass)** — препроцесор для CSS, який дає змогу створювати більш структуровані, модульні стилі, що легко підтримуються. Зокрема, у великому інтерфейсі з численними елементами це дозволяє уникнути конфліктів стилів і забезпечити єдність дизайну.
- **Axios / Fetch API** — інструменти для здійснення HTTP-запитів до серверу. У межах цього проєкту вони застосовуються для взаємодії з API: отримання списку товарів, надсилання даних про замовлення, обробка аутентифікації тощо.
- **ESLint + Prettier** — засоби контролю якості коду. Вони забезпечують єдиний стиль написання, дотримання найкращих практик і своєчасне виявлення помилок на ранньому етапі.

Обґрунтування вибору технологій

Застосування саме React + TypeScript у зв’язці з Redux Toolkit забезпечує:

- **масштабованість інтерфейсу** — зручна структура компонентів, які легко повторно використовуються;
- **керованість станом** — уніфікований підхід до зберігання, оновлення та відстеження даних;
- **типобезпеку** — значно менше помилок під час компіляції та автопідказки в IDE;
- **швидкість розробки** — завдяки готовим шаблонам, утилітам і великій спільноті;
- **легкість інтеграції** з API, бекендом і зовнішніми сервісами (платіжні системи, поштові сервіси тощо).

У підсумку, вибраний технологічний стек дозволяє забезпечити швидкодію, надійність і гнучкість інтерфейсу інтернет-магазину, при цьому зберігаючи високу якість коду та комфорт розробника під час підтримки системи.

РОЗДІЛ 2. СТЕК ТЕХНОЛОГІЙ

React — це популярна бібліотека JavaScript для створення інтерфейсів користувачів. Її основною особливістю є компонентний підхід, що дозволяє розділити додаток на незалежні, багаторазово використовувані частини. Завдяки віртуальному DOM React значно підвищує продуктивність програми, дозволяючи швидко оновлювати лише ті частини інтерфейсу, які змінилися. Ця бібліотека ідеально підходить для створення динамічних SPA-додатків (Single Page Application), таких як інтернет-магазини.

TypeScript — це надбудова над JavaScript, яка додає підтримку статичної типізації. Це означає, що розробники можуть заздалегідь виявляти помилки в коді під час написання. TypeScript покращує читабельність і підтримку проектів, особливо великих додатків із великою кількістю компонентів і логіки. У поєднанні з React ця мова програмування дозволяє створювати зрозумілий, безпечний та легко підтримуваний код.

Redux Toolkit — це офіційний інструмент для спрощення роботи з Redux, який використовується для управління станом програми. Він усуває зайву складність стандартного Redux, автоматизуючи створення дій та ред’юсерів. Redux Toolkit ідеально підходить для зберігання та управління глобальними даними, такими як інформація про користувача, стан кошика покупок, активні фільтри товарів або списки категорій. Завдяки цьому інструменту досягається структурованість коду та полегшується масштабування.

Sass (Syntactically Awesome Stylesheets) є розширенням для CSS, яке надає більш гнучкий і зручний синтаксис для написання стилів. Завдяки Sass можна використовувати змінні, вкладення, міксіни та інші функції, що значно спрощують підтримку великого проекту. Для інтернет-магазину, де кожен компонент має власне стилістичне оформлення, використання Sass забезпечує узгодженість стилів та їх легке оновлення.

Swiper — це сучасна бібліотека для створення адаптивних слайдерів і каруселей. Її можливості дозволяють інтегрувати слайди з товарами, акціями або популярними категоріями. Swiper підтримує сенсорне управління, інтерактивні анімації, автоматичне програвання та адаптивність, що робить його ідеальним вибором для вебдодатків, орієнтованих на мобільних користувачів.

Visual Studio Code (VS Code) — це універсальна та зручна IDE, яку я використовую для роботи з вищезазначеним стеком технологій. Завдяки численним розширенням, таким як ESLint, Prettier та GitLens, середовище стає максимально комфортним для написання чистого та організованого коду.

Node.js є основою для встановлення бібліотек та залежностей. З його допомогою я використовую менеджери пакетів npm або yarn для інтеграції та управління необхідними інструментами.

Для форматування коду використовується **Prettier** із конфігураційним файлом **.prettierrc**, що дозволяє налаштувати правила для табуляції, лапок, ширини рядків та інших аспектів коду. Це сприяє збереженню єдиного стилю в проектах.

CRACO (Create React App Configuration Override) дає змогу кастомізувати конфігурацію Webpack у React-додатках без "eject". З його допомогою реалізую:

- Абсолютні шляхи (@/components, @/assets);
- Підключення плагінів або розширень.

Цей стек забезпечує високий рівень продуктивності, структури та зручності для створення сучасних веб-додатків.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Інтерфейс

Інтерфейс веб-додатка є ключовим елементом для забезпечення ефективної взаємодії користувачів із системою. У цьому розділі детально розглянуто основні компоненти інтерфейсу інтернет-магазину Ecobazar, їхню роль та функціональність.

3.1.1. Структура сторінок

Ecobazar містить кілька основних сторінок, кожна з яких виконує певну функцію, забезпечуючи зручність користування та ефективну навігацію.

Головна сторінка – перше, що бачить користувач при вході на сайт. Вона містить рекламні банери, акційні пропозиції, добірки популярних товарів та навігаційне меню, що дозволяє швидко перейти до необхідного розділу.

Каталог продуктів – містить перелік товарів, згрупованих за категоріями. Для зручності реалізовано систему фільтрів за ціною, брендом, рейтингом, наявністю знижок та іншими параметрами. Також передбачена можливість сортування товарів за популярністю, новизною або вартістю.

Сторінка входу та реєстрації – забезпечує автентифікацію користувачів. На сторінці входу користувач вводить логін та пароль, а на сторінці реєстрації заповнює форму з необхідними даними. Також є можливість авторизації через соціальні мережі.

Кошик – відображає список вибраних товарів із можливістю редагування кількості, видалення позицій та перегляду загальної вартості замовлення. Також на цій сторінці можна застосовувати промокоди та переходити до оформлення замовлення.

Модальні вікна – використовуються для швидкого перегляду інформації без переходу на окремі сторінки.

Наприклад:

- *Модальне вікно швидкого перегляду товару* містить основну інформацію (зображення, назву, ціну, рейтинг та короткий опис), а також кнопку "Додати до кошика".
- *Вікно підтвердження дій*, при видаленні товару з кошика або виході з акаунта.
- *Вікно авторизації*, яке з’являється при спробі оформити замовлення без входу в акаунт.
- *Вікно "Subscribe our newsletter"* – містить поле введення електронної пошти, де користувач може підписатися на розсилку, отримати сповіщення про оновлення та скористатися знижкою 20%.

3.1.2. Навігація

Система навігації забезпечує інтуїтивне переміщення між сторінками та швидкий доступ до основних функцій магазину.

Головне меню – розташоване у верхній частині сторінки та містить посилання на основні розділи: "Головна", "Каталог", "Сторінки", "Блог", "Про нас", "Контакти" та "Кошик".

Пошуковий рядок – дозволяє швидко знаходити товари за ключовими словами. Реалізовано автодоповнення, яке пропонує популярні запити.

Фільтри та сортування – інтегровані в каталог продуктів, допомагаючи користувачам швидко знайти необхідний товар.

Хлібні крихти – відображають шлях користувача до поточної сторінки та спрощують навігацію.

Футер – містить додаткові посилання на інформаційні сторінки, політику конфіденційності та контактні дані.

3.1.3. Карта товару

Кожен товар у каталозі має детальну інформаційну сторінку, яка включає:

1. **Зображення товару** – велике фото з можливістю перегляду галереї.
2. **Галерея фото продукту** – реалізована у вигляді слайдера через Swiper. Ця галерея дозволяє переглядати товар з різних ракурсів, оцінити його зовнішній вигляд, деталі та текстуру. Завдяки інтерактивному слайдеру користувач може легко гортати зображення, збільшувати їх для кращого огляду та отримувати повне уявлення про товар перед покупкою. Це особливо корисно для продуктів, де важливі візуальні характеристики, наприклад, свіжі продукти, одяг або аксесуари.
3. **Назва та короткий опис** – допомагає швидко ознайомитися з основними характеристиками.
4. **Ціна** – відображається поточна ціна, а також знижена (якщо є акція).
5. **Рейтинг та відгуки** – користувачі можуть переглядати оцінки інших покупців та залишати свої відгуки.
6. **Кнопка "Додати в кошик"** – основний елемент взаємодії, що дозволяє швидко додати товар до замовлення.
7. **Додаткова інформація** – характеристики товару (склад, розмір, виробник), умови доставки та оплати.

Залежно від контексту відображення, товари можуть бути представлені у двох форматах — великий товар (*BigProduct*) та малий товар (*SmallProduct*). Кожен з них має свою структуру і набір елементів, які використовуються для ефективного подання інформації.

Слід також зазначити, що стилі окремих елементів можуть відрізнятися відповідно до формату товару. Наприклад, кнопка "Додати в кошик" у великому товарі може бути

більша, помітніша та супроводжуватися додатковими елементами, тоді як у малому варіанті вона більш компактна й інтегрована безпосередньо в картку.

У коді застосунку в окремих частинах сторінки реалізовано умовну логіку, яка визначає, який із компонентів — BigProduct чи SmallProduct — буде використано в конкретному випадку. Це дає змогу адаптувати візуальне подання товару залежно від контексту (наприклад, головна сторінка, сітка популярних товарів чи сторінка самого продукту).

Незважаючи на відмінності у HTML-структурі та стилях оформлення, обидва компоненти мають спільну типову базу: використовується загальний тип у TypeScript — **ProductItem**. Це забезпечує єдиний підхід до опису структури даних товару, спрощує обробку даних, покращує типізацію та уніфікує доступ до властивостей товару в різних частинах проєкту.

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1 Створення структури проекту

Для розробки вебзастосунку ми використовуємо середовище розробки Visual Studio Code (VS Code). Спочатку створюємо структуру проєкту відповідно до сучасних стандартів розробки вебзастосунків.

Перед початком розробки необхідно створити основну папку проекту, у якій будуть розміщені як фронтенд, так і бекенд частини. Оскільки даний проект зосереджений на розробці клієнтської частини, далі розглянемо папку **client**, де буде зберігатися весь код для інтерфейсу користувача.

Згідно з обраним дизайном, який попередньо був розроблений у Figma, ми визначаємо основні компоненти та файли, які будуть використані у проєкті. Структура проєкту організована наступним чином:

client
├── node_modules/
├── public/
├── src/
│ ├── assets/
│ │ ├── fonts/
│ │ ├── images/
│ ├── components/
│ ├── consts/
│ ├── data/
│ ├── hooks/
│ ├── pages/
│ ├── styles/
│ ├── types/
│ ├── utils/
│ ├── App.tsx

```
│   ├── index.tsx
│   ├── styles.d.ts
├── package.json
├── tsconfig.json
├── vite.config.ts
```


Пояснення основних папок та файлів:

- **node_modules/** – директорія, яка містить усі встановлені npm-залежності.
- **public/** – папка для статичних файлів, таких як favicon, HTML-шаблон та інші ресурси.
- **src/** – головна папка з вихідним кодом проєкту.
- **assets/** – містить додаткові ресурси, такі як шрифти та зображення.
- **components/** – містить багаторазові UI-компоненти.
- **consts/** – містить файли з константами для проєкту.
- **context/** – містить контексти React для глобального стану.
- **data/** – містить локальні тестові дані.
- **hooks/** – містить кастомні React-хуки.
- **pages/** – містить основні сторінки застосунку.
- **styles/** – містить глобальні стилі та SCSS-файли.
- **themes/** – містить теми оформлення.
- **types/** – містить файли з TypeScript-типами.
- **utils/** – містить допоміжні утиліти та функції.
- **App.tsx** – головний компонент застосунку.
- **index.tsx** – точка входу в застосунок.
- **styles.d.ts** – містить описи стилів для TypeScript.
- **package.json** – містить інформацію про залежності та налаштування проєкту.
- **package-lock.json** – файл для точного керування версіями залежностей.
- **craco.config.js** – конфігураційний файл для Craco (налаштування Webpack).
- **TODO.tsx** – файл із завданнями для розробки.
- **tsconfig.json** – конфігураційний файл TypeScript.

4.2. Ініціалізація проєкту та встановлення залежностей

Перед початком роботи над клієнтською частиною необхідно ініціалізувати проєкт та встановити всі необхідні залежності. Використовується пакетний менеджер **npm** для керування бібліотеками та їх версіями.

Ініціалізація проєкту

Спочатку потрібно перейти в основну папку ecobazar і створити фронтенд-частину проєкту:

npx create-react-app ecobazar --template typescript

У результаті автоматично створилася базова структура React-проєкту з підтримкою TypeScript, яка включає такі основні елементи:

- **src/** — папка з вихідним кодом
- **public/** — публічні ресурси
- **tsconfig.json** — файл конфігурації TypeScript
- **package.json** — файл із переліком залежностей і скриптів

Налаштування абсолютних імпортів

Для зручності імпорту файлів у великих проєктах рекомендується використовувати абсолютні шляхи, щоб уникнути довгих відносних шляхів по типу ../../../. Для цього було відредаговано файл tsconfig.json, додавши наступні шляхи у секцію **compilerOptions.paths**:

```
"paths": {
  "@/*": ["src/*"],
  "@/styles/*": ["src/styles/*"],
  "@images/*": ["src/assets/images/*"],
  "@/components/*": ["src/components/*"],
  "@/pages/*": ["src/pages/*"]
}
```

Це дозволяє виконувати імпорти типу:

```
import Header from "@/components/Header";
import "@/styles/global.scss";
```

Встановлення необхідних залежностей

Після створення структури були встановлені залежності, необхідні для реалізації основного функціоналу застосунку. Основні бібліотеки та їх призначення:

- **@craco/craco** — для можливості редагування конфігурації CRA без eject.
- **react-router-dom** — маршрутизація між сторінками.
- **react-redux, @reduxjs/toolkit** — глобальний стан застосунку.
- **axios** — HTTP-запити до API.
- **classnames** — умовне підключення класів CSS.
- **sass** — стилізація з використанням SCSS.
- **swiper** — реалізація слайдерів/каруселей.
- **prettier** — автоформатування коду.

Розробницькі залежності (devDependencies):

- **typescript** — мова, що забезпечує типізацію.
- **@types/*** — типи для бібліотек.
- **@testing-library/*** — бібліотеки для тестування компонентів.

Усі зазначені залежності були встановлені командою **npm install.**

Ця команда читає конфігураційний файл package.json, який містить повний перелік як основних (dependencies), так і розробницьких (devDependencies) бібліотек із відповідними версіями. На основі цієї інформації система автоматично завантажує необхідні пакети та розміщує їх у спеціальну службову директорію node_modules.

Також створюється файл package-lock.json, який фіксує точні версії встановлених пакетів, забезпечуючи стабільність середовища при подальшому розгортанні або передачі проєкту.

Якщо ж файл package.json відсутній (наприклад, у випадку створення нового проєкту з нуля без використання шаблонів), перед встановленням залежностей його необхідно створити вручну командою **npm init** або для швидкого створення з типових налаштувань **npm init -y.**

Після цього для кожної окремої бібліотеки необхідно виконати встановлення вручну, наприклад:

npm install react-router-dom
npm install @reduxjs/toolkit react-redux

А для додавання типів у TypeScript-проєкті використовують:
npm install --save-dev @types/react-router-dom

Таким чином, наявність коректного package.json значно спрощує процес підготовки робочого середовища та гарантує, що всі учасники проєкту будуть працювати з однаковими версіями бібліотек.

Після успішного встановлення залежностей проєкт готовий до подальшого налаштування та початкової розробки.

4.3. Початкове налаштування структури проєкту

Після встановлення всіх необхідних залежностей було виконано базове налаштування структури клієнтської частини застосунку. На цьому етапі відбувалося очищення

стартового коду, оновлення конфігураційних файлів та підготовка проєкту до подальшої розробки інтерфейсу користувача.

Очищення шаблонного коду

Після створення проєкту за допомогою команди *npx create-react-app ecobazar --template typescript* у проєкті автоматично генерується шаблонна структура, яка містить демонстраційні файли та непотрібні для реального застосунку приклади.

Для забезпечення чистої та організованої основи було видалено такі файли з папки src/:

- **App.test.tsx** — тестовий файл для компонента App;
- **logo.svg** — демонстраційне зображення логотипу React;
- **reportWebVitals.ts** — скрипт для вимірювання продуктивності;
- **setupTests.ts** — конфігураційний файл для тестування.

Додатково були очищені такі файли:

- **App.tsx** — видалено початкову розмітку та логіку, залишено лише базову функцію компонента;
- **App.css** — стилі були або повністю очищені, або перенесені у формат SCSS для подальшої роботи з препроцесором;
- **index.tsx** — видалено виклик функції reportWebVitals(), а також оновлено структуру для підключення маршрутизації та глобального стану (див. нижче).

Оновлення index.tsx

У файлі src/index.tsx були додані наступні зміни:

1. Імпортовано **BrowserRouter** з бібліотеки react-router-dom для забезпечення маршрутизації між сторінками;
2. Імпортовано **Provider** з react-redux та підключено store для глобального стану;

3. Імпортовано основні глобальні стилі, написані на SCSS (наприклад, @import "@/styles/index.scss");

Вся структура компонента App була обгорнута у відповідні провайдери.

Це дозволило з самого початку задати коректну архітектуру застосунку для зручного масштабування.

Налаштування alias-імпортів

Для зручності навігації та уникнення довгих відносних шляхів (наприклад, ../../../components) було оновлено конфігураційний файл tsconfig.json. Це значно полегшує імпорти у коді та забезпечує зручність для розробників у роботі з великими проєктами. У секцію **compilerOptions** додано параметр paths, що дозволяє створити псевдоніми для основних директорій:

```
"paths": {
  "@/*": ["src/*"],
  "@/styles/*": ["src/styles/*"],
  "@images/*": ["src/assets/images/*"],
  "@/components/*": ["src/components/*"],
  "@/pages/*": ["src/pages/*"]
```

Очищення стилів та підготовка SCSS

Паралельно було прийнято рішення про використання препроцесора Sass. Для цього було встановлено відповідну залежність (sass) і створено базову структуру SCSS-файлів у папці src/styles. Також було видалено файли App.css та index.css, оскільки їхня функціональність була повністю перенесена до більш гнучкої та масштабованої SCSS-структури.

Цей підхід дозволив централізувати всі базові стилістичні налаштування, спростити подальший супровід проєкту та забезпечити повторне використання змінних,

міксинів та інших можливостей, які надає Sass. Структура папки styles була організована з урахуванням майбутнього масштабування інтерфейсу, поділу логіки оформлення та збереження чистоти коду.

На момент налаштування проєкту ця структура виглядала наступним чином:

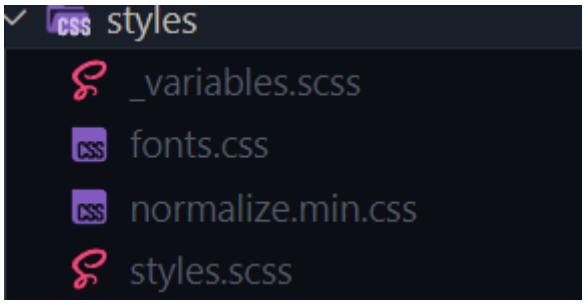

Рисунок 4.1. Структура папки styles

_variables.scss

Файл _variables.scss містить набір SCSS-змінних, які використовуються по всьому проєкту. Тут зосереджено основні кольори, типографічні параметри, змінні для відступів, брейкпоінтів, радіусів скруглення, розмірів шрифтів тощо. Такий підхід дозволяє легко змінювати зовнішній вигляд усього застосунку через редагування одного централізованого файлу. Це особливо важливо при роботі з темами або при редизайні.

Завдяки використанню змінних вдається уникнути дублювання коду та підвищити його гнучкість, підтримуючи принцип **DRY (Don't Repeat Yourself).**

fonts.css

У цьому файлі реалізовано підключення шрифтів, які використовуються у проєкті. Оскільки формат CSS дозволяє зручно підключати шрифти як із зовнішніх джерел (наприклад, Google Fonts), так і локальні за допомогою @font-face, було вирішено залишити саме .css, не перетворюючи його на SCSS.

У майбутньому можливе винесення налаштувань шрифтів у SCSS або створення додаткових файлів для розширення типографіки, але на початковому етапі файл fonts.css виконує свою функцію достатньо ефективно.

normalize.min.css

Цей файл є мінімізованою версією бібліотеки Normalize.css, яка забезпечує однаковий базовий вигляд HTML-елементів у всіх популярних браузерах. Це необхідно, щоб уникнути відмінностей у відображенні елементів, які можуть виникати через різні стилі за замовчуванням у Chrome, Firefox, Safari та інших.

Підключення Normalize.css на самому початку стилів дає змогу вирівняти початковий стан DOM і розробляти інтерфейс з нуля, без непередбачених впливів від нативних стилів браузера.

styles.scss

Це головний SCSS-файл, що агрегує всі глобальні стилі, які мають використовуватись у застосунку. Він виконує роль єдиної точки входу для стилістичних ресурсів і саме через нього підключаються normalize, шрифти та змінні. Це дозволяє впорядковано організувати структуру імпорту стилів і забезпечити зрозумілу логіку розташування стилістичного коду. Типовий вміст styles.scss виглядає наступним чином:

```scss
@import './normalize.min.css';
@import './_variables.scss';
@import './fonts.css';

body {
    font-family: 'Poppins', sans-serif;
}

button,
input,
textarea {
    font-family: 'Poppins', sans-serif;
}

input {
    &[type="file"],
    &[type="checkbox"] {
        display: none;
    }
}

img {
    user-select: none;
}
```

Рисунок 4.2 – Головний SCSS-файл

Завдяки цій централізації підтримка стилів проєкту значно спрощується, адже достатньо відкрити лише один файл для внесення більшості глобальних змін.

Підключення стилів у index.tsx

Після структурування SCSS-файлів, головний файл styles.scss було підключено у src/index.tsx наступним чином:

import '@/styles/styles.scss';

Використання alias-імпорту (@/styles/...) стало можливим завдяки попередньо налаштованому шляху в tsconfig.json. Це дозволяє уникнути надмірного використання відносних шляхів типу ../../styles/styles.scss, що позитивно впливає на читабельність та зменшує ризик помилок при переміщенні файлів.

4.4. Навігація та маршрутизація

Для організації навігації між сторінками у застосунку використовується бібліотека **React Router DOM**, яка забезпечує маршрутизацію у SPA (Single Page Application) форматі. Основним компонентом маршрутизації є BrowserRouter, який дозволяє слідкувати за змінами URL-адреси без повного перезавантаження сторінки.

Підключення маршрутизатора

У файлі index.tsx уся структура застосунку обгорнута у компонент <BrowserRouter>, а також у компонент <Provider>, який передає Redux-сховище:

```
root.render(
    <Provider store={store}>
        <BrowserRouter>
            <App />
        </BrowserRouter>
    </Provider>
);
```

Рисунок 4.3 – Маршрутизатор

Підключення Redux Store

У цьому проєкті для керування глобальним станом використовується **Redux Toolkit**. Store підключається у кореневому файлі index.tsx за допомогою компонента <Provider> з пакету react-redux. Це забезпечує доступ до Redux-сховища у всіх вкладених компонентах. Функція **setupStore**() створює Redux Store і знаходиться у файлі *store/index.ts*. Нижче представлений її код:

```
const reducers = {
    // ...
};

const rootReducer = combineReducers({
    ...reducers
});

export const setupStore = () => {
    return configureStore({
        reducer: rootReducer
    });
};

export type RootState = ReturnType<typeof rootReducer>;
export type AppStore = ReturnType<typeof setupStore>;
export type AppDispatch = AppStore['dispatch'];
export default setupStore;
```

Рисунок 4.4 – Підключення Redux Store

Структура Store

store/

├── actions/ – (опціонально) окремі файли з діями

├── reducers/ – окремі slice-и для різних частин стану

│ ├── userSlice.ts – стан, пов’язаний із користувачем (авторизація тощо)

│ └── counterSlice.ts – демонстраційний лічильник

└── index.ts – об’єднання ред’юсерів і створення Store

Авторизація

У слайсі **userSlice** зберігається булеве значення isAuth, яке означає, чи є користувач авторизованим. На початковому етапі isAuth зберігається лише у Redux Store, тобто визначається на клієнтській стороні. У майбутньому передбачається перевірка авторизації через бекенд — наприклад, після реєстрації або логіну буде виконуватись запит на сервер для підтвердження сесії.

Типізація Redux у TypeScript

Для забезпечення зручності та повторного використання, ми створюємо кастомні хуки **useAppDispatch** та **useAppSelector**:

```
import { useDispatch, useSelector } from 'react-redux';  3.9k (gzipped: 1.7k)
import type { RootState, AppDispatch } from '../store';

export const useAppDispatch = useDispatch.withTypes<AppDispatch>();
export const useAppSelector = useSelector.withTypes<RootState>();
```

Рисунок 4.5 – Кастомні типізовані хуки

- **useAppSelector** — це типізована версія useSelector, яка дозволяє TypeScript точно знати, який тип має state. Це забезпечує автодоповнення та уникнення помилок.
- **useAppDispatch** — типізована версія useDispatch, яка дозволяє передавати у dispatch лише ті екшени, які очікує наш AppStore.

Обидва ці хуки створені один раз і використовуються у всіх компонентах, що робить код чистішим, безпечнішим та зручнішим для підтримки.

Компонент AppRouter

Основну логіку маршрутизації реалізовано в окремому компоненті AppRouter, який визначає, які сторінки може переглядати користувач залежно від того, чи він авторизований.

Іншими словами, саме цей компонент керує тим, що бачить користувач — наприклад, якщо він не ввійшов у систему, то йому доступні лише базові сторінки на кшталт головної чи реєстрації. Після входу відкривається доступ до основного функціоналу додатка, зокрема до сторінки з товарами.

```
const AppRouter: FC = () => {
    const { isAuth } = useAppSelector(state => state.user);

    return (
        isAuth ?
            <Routes>
                {authRoutes.map(route =>
                    <Route
                        key={route.path}
                        path={route.path}
                        element={<route.element />}
                    />
                )}
                <Route path='*' element={<Navigate to={RouteNames.ERROR_ROUTE} replace />} />
            </Routes>
            :
            <Routes>
                {guestRoutes.map(route =>
                    <Route
                        key={route.path}
                        path={route.path}
                        element={<route.element />}
                    />
                )}
                <Route path='*' element={<Navigate to={RouteNames.LOGIN_ROUTE} replace />} />
            </Routes>
    )
}
export default AppRouter;
```

Рисунок 4.6 – Компонент AppRouter

Цей компонент отримує значення **isAuth** зі стану Redux за допомогою кастомного хука useAppSelector. Залежно від цього значення, застосунок визначає, які маршрути потрібно відображати: маршрути для авторизованих користувачів (authRoutes) або маршрути для неавторизованих користувачів (guestRoutes). Якщо введена адреса не відповідає жодному з визначених маршрутів, користувача автоматично перенаправляє на сторінку помилки або сторінку входу.

Цей компонент отримує значення isAuth зі стану Redux за допомогою кастомного хука useAppSelector. Саме це значення вказує, чи користувач є авторизованим. Залежно від нього застосунок динамічно визначає, які маршрути слід використовувати:

- **authRoutes** – список маршрутів, доступних для авторизованих користувачів (наприклад, домашня сторінка, сторінка магазину, тощо).
- **guestRoutes** – маршрути для неавторизованих користувачів (наприклад, сторінка входу).

Це дозволяє обмежити доступ до певних сторінок лише для зареєстрованих або авторизованих користувачів, забезпечуючи тим самим захист особистих даних та контрольований доступ до функціоналу.

Крім того, у AppRouter реалізовано обробку помилкових маршрутів: якщо користувач введе адресу, яка не відповідає жодному з наявних маршрутів, він буде перенаправлений на сторінку помилки (/error) або на сторінку входу (/login) — залежно від статусу авторизації. Це гарантує зручність навігації та запобігає потраплянню на "биті" або порожні сторінки.

Файл routes.ts: централізоване управління маршрутами

Для зручного управління маршрутами у проєкті створено окремий файл routes.ts, який зберігається в папці consts. У ньому оголошено перелік маршрутів за допомогою типів і переліку констант. Такий підхід дозволяє:

- централізовано керувати всіма шляхами сторінок;
- уникати дублювання значень шляхів у різних частинах коду;
- забезпечити масштабованість, особливо в великих проектах.

У прикладі вище спочатку імпортуються компоненти сторінок, які відповідають за вміст відповідних маршрутів. Далі оголошується інтерфейс Route, що описує структуру кожного маршруту – шлях (path) та компонент (element), який буде відображено.

Також створено enum RouteNames – перелік зрозумілих іменованих маршрутів, який замінює строкові значення (на кшталт '/login') і дозволяє легко посилатися на шляхи у всьому проєкті без ризику помилки.

Два масиви — guestRoutes та authRoutes — формують доступні шляхи залежно від статусу користувача. Саме вони імпортуються та використовуються в компоненті AppRouter при рендерінгу маршрутизатора.

```
export enum RouteNames {
    HOME_ROUTE = '/',
    SHOP_ROUTE = '/shop',
    LOGIN_ROUTE = '/login',
    LOGOUT_ROUTE = '/logout',
    REGISTER_ROUTE = '/register',
    ERROR_ROUTE = '/error'
}

export const guestRoutes: Route[] = [
    {
        path: RouteNames.LOGIN_ROUTE,
        element: Login
    },
];

export const authRoutes: Route[] = [
    {
        path: RouteNames.HOME_ROUTE,
        element: Home
    },
    {
        path: RouteNames.SHOP_ROUTE,
        element: Shop
    },
    {
        path: RouteNames.ERROR_ROUTE,
        element: Error
    },
];
```

Рисунок 4.7 – Масиви маршрутів

Статус авторизації (isAuth)

На поточному етапі розробки змінна isAuth є частиною локального стану Redux. Вона оновлюється після входу або реєстрації користувача і визначає, які сторінки мають бути доступними.

У подальшому передбачається, що перевірка авторизації здійснюватиметься через бекенд — наприклад, за допомогою JWT-токена або сесійної автентифікації. Це дозволить після оновлення сторінки або повторного відкриття застосунку автоматично визначити статус користувача на основі відповіді сервера. Таке рішення забезпечить безпечну та стабільну систему авторизації, незалежно від локального стану або перезавантаження сторінки.

4.5. Компоненти

Після налаштування основи проекту настав момент поступово переходити до створення інтерфейсу. У React практично все зводиться до компонентів — від простих кнопок до цілих секцій сторінки. Саме тому на цьому етапі основний фокус було зосереджено на розробці гнучких, зручних у використанні та повторному застосуванні компонентів.

4.5.1. Основні компоненти

Головним компонентом усього застосунку є **App.tsx.** Саме з нього починається побудова інтерфейсу користувача. У фінальному вигляді компонент **App** має наступну структуру:

```tsx
const App: FC = () => {
    return (
        <>
            <Header />
            <AppRouter />
            <Footer />
        </>
    )
}
export default App;
```

Рисунок 4.8 – Компонент App

Цей компонент відображає шапку (**Header**), маршрутизацію сторінок через AppRouter, а також нижню частину інтерфейсу (**Footer**). Такий підхід дозволяє централізовано керувати загальним макетом сторінки, відокремивши загальні елементи від змінного вмісту (який керується маршрутизацією).

Компонент Header

Компонент Header виконує роль шапки сайту та складається з трьох основних частин:

- TopBar — верхній інформаційний рядок (наприклад, мова, підтримка).
- MainHeader — головний блок з логотипом і пошуком.
- Navbar — навігаційне меню для переходу між сторінками.

```tsx
const Header: FC = () => {
    return (
        <header className={styles.header}>
            <TopBar />
            <MainHeader />
            <Navbar />
        </header>
    )
}
export default Header;
```

Рисунок 4.9 – Компонент Header

Компонент Footer

Компонент Footer відповідає за нижню частину сторінки та містить таку інформацію:

- логотип;
- контактні дані (телефон та електронна пошта);
- навігаційні посилання, згруповані за категоріями;
- інформацію про авторські права;іконки платіжних систем (наприклад, Visa, Mastercard, Discover).

```
const Footer: FC = () => {
    return (
        <footer className={styles.footer}>
            <Container>
                <div className={styles.footerTop}>···
                </div>
                <div className={styles.footerBottom}>···
                </div>
            </Container>
        </footer>
    )
}
export default Footer;
```

Рисунок 4.10 – Компонент Footer

Компонент структурований таким чином, щоб підтримувати масштабованість — додавання нових категорій або контактних каналів не вимагає змін логіки, а лише оновлення вхідних даних (footerLinks, paymentCards). Усі стилі винесені в окремий SCSS-модуль Footer.module.scss.

Компонент Hero

Компонент Hero відповідає за відображення головного блоку на головній сторінці сайту. Це перший візуальний елемент, з яким взаємодіє користувач після завантаження сторінки. Основна мета цього компонента — ознайомити користувача з ресурсом, звернути увагу на головні переваги магазину, а також надати коротке і привабливе повідомлення про послуги або акції.

Цей компонент складається з двох ключових частин — рекламного банера та блоку з перевагами. Обидва елементи відображаються всередині компонента-контейнера, що забезпечує правильне вирівнювання та адаптивність вмісту.

```
const Hero: FC = () => {
    return (
        <div className={styles.hero}>
            <Container>
                <Banner />
                <Features />
            </Container>
        </div>
    )
}
export default Hero;
```

Рисунок 4.11 – Компонент Hero

Опис структури компонента

Компонент Hero є функціональним компонентом, який повертає TSX-розмітку. Основні елементи в його структурі:

Обгортка **div** з класом **hero** — відповідає за зовнішній вигляд блоку. Сам стиль визначається у відповідному SCSS-файлі, де задаються відступи, фон, поведінка при різних розмірах екрану тощо.

<**Container**> — допоміжний компонент, що вирівнює вміст по центру та обмежує його максимальну ширину. Це дозволяє забезпечити зручне читання й гармонійну структуру сторінки.

<**Banner** /> — окремий компонент, що виводить заголовки, рекламні слогани, зображення або кнопки із закликом до дії.

<**Features** /> — блок із ключовими перевагами магазину. Зазвичай у цьому компоненті вказуються вигідні умови доставки, акційні пропозиції, гарантії якості продукції тощо.

Увесь компонент Hero працює як незалежний модуль, який легко підтримувати, змінювати або перевикористовувати.

Компонент CategoryCards

Компонент відображає блок з популярними категоріями товарів на головній сторінці. Його головне завдання — дати користувачу швидкий доступ до основних товарних груп, що покращує навігацію по сайту та підвищує зручність використання.

Компонент реалізований як функціональний та використовує масив з даними категорій, де кожна категорія має зображення, назву та унікальний ідентифікатор. Дані імпортуються з окремого файлу.

Усередині компонента використовуються:

- Обгортка div з класом categories — для загального стилю.
- Компонент **Container** — обмежує ширину та вирівнює вміст.
- Компонент **Title** — виводить заголовок «Popular Categories».

Мапінг масиву категорій — для кожного елемента показується зображення та назва.

Стилі написані через SCSS-модулі, що дозволяє уникнути конфліктів та зробити компонент адаптивним.

Загалом, компонент CategoryCards — це зручний та візуально привабливий блок, який допомагає користувачу швидше знайти потрібний розділ сайту.

Компонент PopularProducts

Компонент має особливу функцію в структурі головної сторінки — він не просто відображає товари, а виконує роль інтерактивного блоку з можливістю швидкої взаємодії з продуктами.

Значення компонента

На відміну від статичних елементів інтерфейсу, цей блок реалізує першу справжню динаміку — користувач може не лише переглянути товари, а й взаємодіяти з ними через функцію швидкого перегляду (Quick View). Це робить компонент важливою частиною UI/UX, особливо в e-commerce-проєктах, де швидкий доступ до деталей товару критично важливий.

Роль onQuickView

Функція onQuickView — це універсальний callback, який передається до кожного окремого продукту у вигляді пропсу. Вона виконує роль обгортки над логікою відкриття модального вікна або панелі з детальною інформацією про товар. Таким чином, кожен товар у списку може бути "активований" — тобто натиснутий, щоб розгорнути швидкий перегляд.

Ця функція винесена "зовні", тобто реалізація не знаходиться всередині самого компонента PopularProducts. Завдяки цьому ми можемо повторно використовувати її в інших частинах сайту, де потрібно викликати швидкий перегляд.

Такий підхід є прикладом передачі функціоналу через пропси — хороша практика в React, бо дозволяє компоненти робити максимально гнучкими і повторно використовуваними.

Вибірка продуктів через **slice**

У компоненті ми не виводимо всі товари, що зберігаються у фронтенд-структурі products, а лише перші десять з них. Це реалізовано за допомогою методу slice(0, 10), який обрізає масив:

- Зменшує навантаження на сторінку;
- Дозволяє фокусувати увагу користувача на популярних товарах;
- Відкриває можливість майбутнього розширення через пагінацію або кнопку "показати більше".

Компонент BannerCards

Компонент відповідає за відображення рекламних банерів на головній сторінці. Це яскравий візуальний блок, який привертає увагу користувача до спеціальних пропозицій, знижок або сезонних акцій. Банери є частиною маркетингового інтерфейсу, покликаного підвищити конверсію.

```
{bannerCards.map(banner =>
    <div
        key={banner.id}
        className={classNames(
            styles.banner,
            { [styles.bannerDark]: banner.isDarkStyle },
            'bg-image'
        )}
        style={{ backgroundImage: `url(${banner.bgImage})` }}
    >
        <p className={styles.bannerLabel}>
            {banner.label}
        </p>
        <h3 className={styles.bannerTitle}>
            {banner.title}
        </h3>
        {banner.text &&
            <p className={styles.bannerText}>
                {banner.text}
                {banner.price ?
                    <span className={styles.bannerPrice}>
                        {banner.price}
                    </span>
                    :
                    <span className={styles.bannerDiscount}>
                        {banner.discount}
                    </span>
                }
            </p>
        }
        {banner.isTimer &&
            <Timer
                customStyles={styles}
                targetDate={targetTimes.saleOfTheMonth}
            />
        }
```

Рисунок 4.12 – Компонент BannerCards

Гнучкість компонента через пропси

Особливість компонента в тому, що він динамічно будується на основі даних із масиву bannerCards, де кожен об'єкт містить повний набір властивостей, які впливають на зовнішній вигляд і логіку окремого банера. Усі ключові параметри передаються через пропси, що дозволяє легко налаштовувати кожен банер окремо:

- **bgImage** — шлях до фонової картинки, яка задається через style={{ backgroundImage: ... }}. Це дозволяє виводити унікальний фон для кожного банера.
- **label** — коротка позначка (наприклад, «New» або «Sale»), яка вказується над заголовком.
- **title** — основний заголовок банера, зазвичай — назва акції.
- **text** — додатковий опис або інформаційне повідомлення.
- **price** та **discount** — взаємовиключні пропси. Якщо передано price, то виводиться ціна; якщо discount — тоді відображається знижка.
- **isDarkStyle** — булеве значення, яке визначає, чи потрібно застосовувати темне оформлення (темний шрифт, напівпрозорий фон тощо).
- **isTimer** — булевий прапорець, що вмикає або вимикає таймер для банера.

Цей підхід дозволяє створювати універсальні банери, які відрізняються між собою лише значеннями пропсів.

У деяких банерах використовується компонент **Timer**, який активується за умови, якщо isTimer: true.

Компонент Timer

Компонент Timer реалізує зворотний відлік часу до певної дати та використовується на сайті для **показу залишку часу до завершення акції**. Така функціональність є важливою частиною інтерфейсу інтернет-магазину, оскільки створює відчуття терміновості та мотивує користувачів зробити покупку.

Компонент приймає два параметри:

- **targetDate** — дата та час завершення акції;
- **customStyles** (необов’язково) — об’єкт зі стилями для можливості гнучкого налаштування зовнішнього вигляду.

```jsx
return (
    <div className={classNames(styles.timer, customStyles?.timer)}>
        {timeUnits.map((unit, index) =>
            <div
                key={unit.label}
                className={classNames(styles.timerItem, customStyles?.timerItem)}
            >
                <div className={classNames(styles.timerUnit, customStyles?.timerUnit)}>
                    <span className={classNames(styles.timerValue, customStyles?.timerValue)}>
                        {unit.value}
                    </span>
                    <p className={classNames(styles.timerLabel, customStyles?.timerLabel)}>
                        {unit.label}
                    </p>
                </div>
                {index < timeUnits.length - 1 && (
                    <div className={classNames(styles.timerColon, customStyles?.timerColon)}>
                        <span></span>
                        <span></span>
                    </div>
                )}
            </div>
        )}
    </div>
)
```

Рисунок 4.13 – Компонент Timer

Компонент візуально складається з чотирьох частин, які відображають кількість днів, годин, хвилин та секунд до завершення акції. Кожен елемент показано у вигляді окремого блоку, що містить числове значення та відповідний підпис. Для зручнішого сприйняття між блоками розміщено декоративні двокрапки, які стилістично розділяють частини таймера.

Оформлення компонента реалізовано за допомогою CSS-модуля **Timer.module.scss**, що дозволяє ізолювати стилі та уникати конфліктів. Крім того, у компоненті передбачено можливість налаштування зовнішнього вигляду через об’єкт **customStyles**, який передається як пропс. Це дає змогу гнучко адаптувати таймер під різні частини інтерфейсу без дублювання коду.

Завдяки простій структурі, автоматичному оновленню значень щосекунди та можливості гнучкого налаштування компонент є зручним у використанні та повторному застосуванні. Його легко адаптувати до будь-якого макету, що робить

Timer хорошим прикладом ефективного застосування React для створення динамічних, підтримуваних і візуально привабливих інтерфейсних елементів.

Кнопка переходу

У кожному банері також використовується кнопка ShopButton, якій передаються пропси background та size. Це дозволяє змінювати зовнішній вигляд кнопки залежно від стилю банера, забезпечуючи візуальну гармонію.

```
const ShopButton: FC<ShopButtonProps> = ({ background = 'transparent', size }) => {
    const buttonClass = classNames(
        styles.button,
        background !== 'transparent' && styles.bg,
        background && styles[`bg-${background}`],
        size ? styles[`button-${size}`] : ''
    );

    return (
        <Link
            to={RouteNames.SHOP_ROUTE}
            className={buttonClass}
        >
            Shop now
            <GreenArrow className={styles.arrow} />
        </Link>
    )
}
export default ShopButton;
```

Рисунок 4.14 – Кнопка переходу

Компонент HotDeals

Компонент HotDeals відповідає за виведення секції акційних товарів (гарячих пропозицій) на головній сторінці інтернет-магазину. Візуально ця секція поділена на дві частини: великий акційний товар та кілька звичайних товарів меншого розміру.

Секція побудована на основі масиву products, який містить усі товари. За допомогою методу reduce відбувається відбір головного акційного товару, який буде відображений великим банером.

Перевіряється перший товар, у якого є позначка isSale: true. Саме він і стає головним продуктом секції (великий банер). Усі інші товари додаються до окремого масиву **remainingProducts**.

Це забезпечує динамічний вибір головної пропозиції без необхідності вручну вказувати товар у коді.

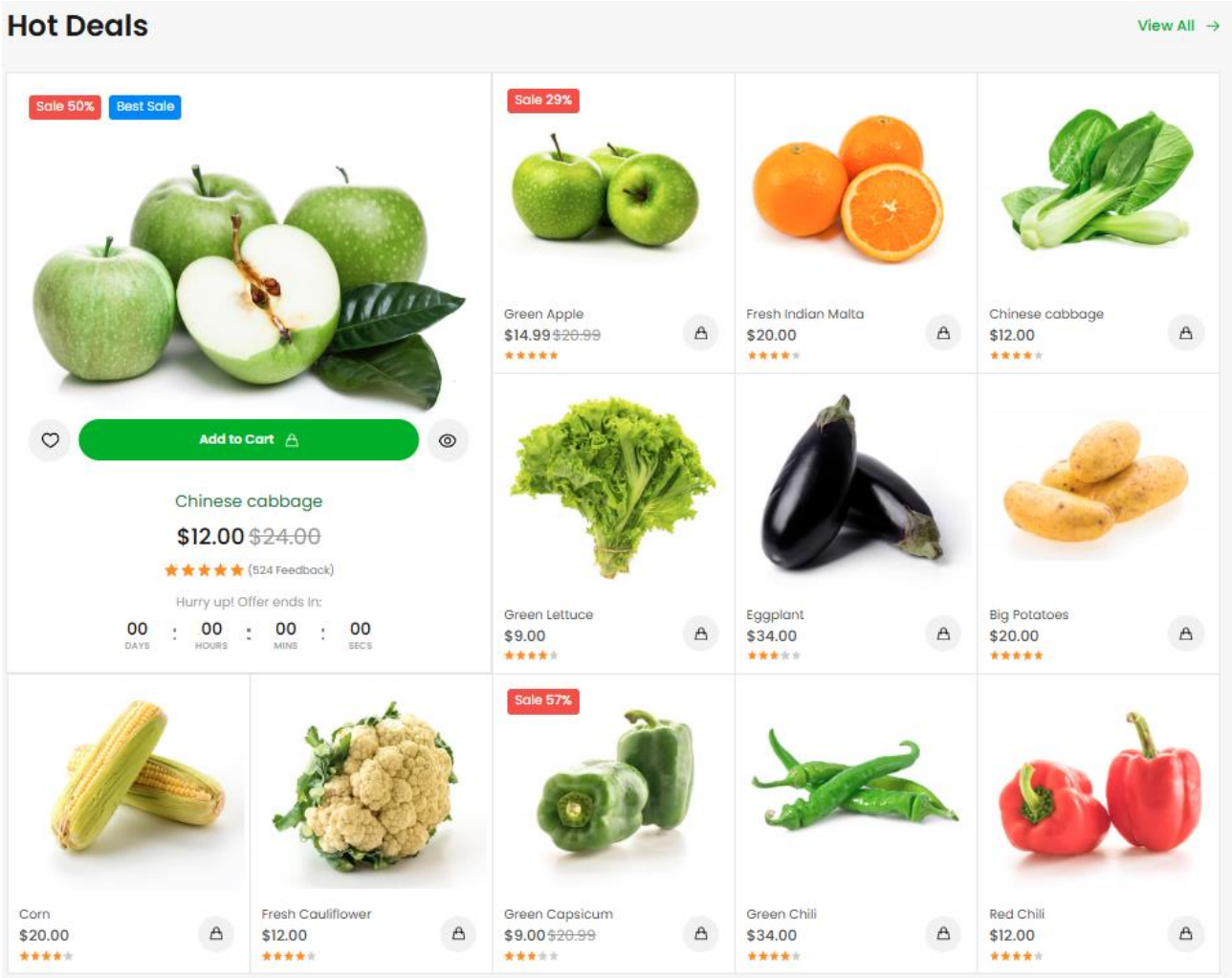

Рисунок 4.15 – Компонент Hot Deals

У компоненті використано два підкомпоненти:

- **BigProduct** — показує головний товар із яскравими елементами, великою кнопкою «Add to cart», таймером (якщо є) і знижкою. Його задача — максимально привернути увагу.
- **SmallProduct** — виводить інші товари меншим розміром, до 10 одиниць, у вигляді сітки.

Усі продукти, як великі, так і малі, отримують функцію onQuickView як пропс — це дозволяє відкривати модальне вікно із деталями товару, тобто реалізована підтримка швидкого перегляду.

Компонент News

Компонент News відповідає за виведення останніх новин на головній сторінці сайту. Його основна задача — інформувати користувача про актуальні події або оновлення, а також створити додатковий інтерес до контенту магазину.

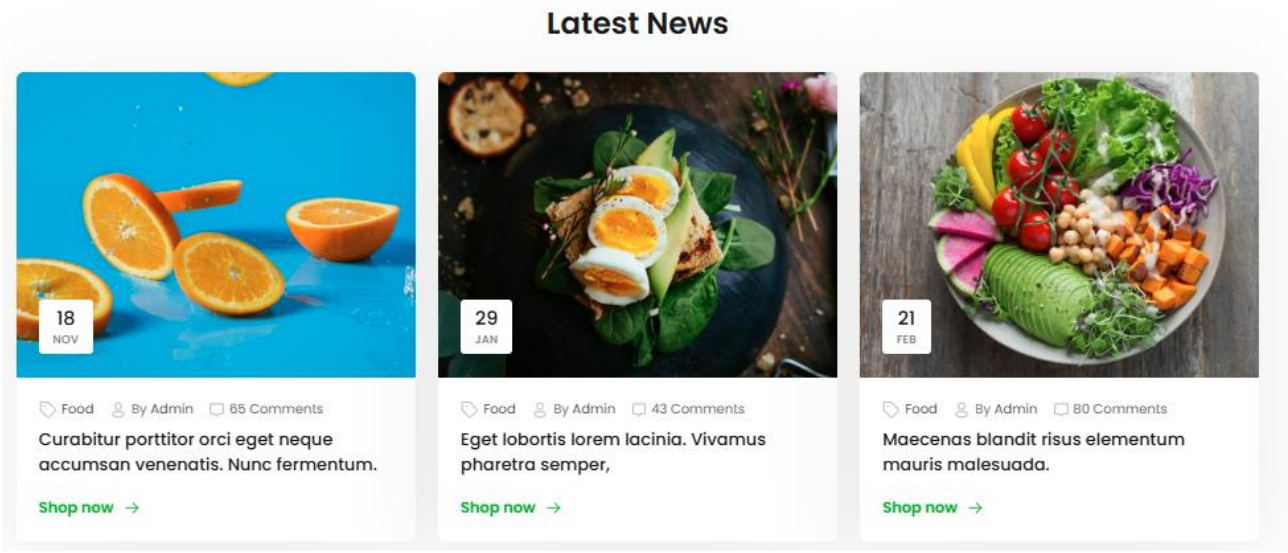

Рисунок 4.16 – Компонент News

Для кожного елемента списку рендериться підкомпонент **NewsItem**, який містить візуальний блок із зображенням, датою (день і скорочений місяць), метаданими та кнопкою переходу. Метадані супроводжуються тематичними іконками — це робить подачу інформації більш структурованою та зручною для сприйняття. Крім того, для всіх зображень автоматично генеруються текстові альтернативи (alt), що покращує доступність інтерфейсу.

Увесь контент загортається в компонент Container, що слугує для вирівнювання та обмеження ширини блоку. Стилі реалізовано через CSS-модулі, що дозволяє уникнути конфліктів класів та зберегти охайність коду.

Блок новин виконує не лише інформативну функцію, а й створює відчуття живого, оновлюваного ресурсу. Він підсилює динаміку сторінки, формує враження актуальності контенту та сприяє зростанню довіри користувачів до сайту.

```
<div className={styles.news}>
    <Container>
        <h2 className={`${styles.title} title`}>
            Latest News
        </h2>
        <div className={styles.container}>
            {news.map(item =>
                <NewsItem
                    key={item.id}
                    item={item}
                />
            )}
        </div>
    </Container>
</div>
```

Рисунок 4.17 – Виведення блоку новин

Компонент Testimonials

Компонент відповідає за виведення блоку з відгуками у вигляді слайдера.

Рисунок 4.18 – Компонент Testimonials

Він реалізований за допомогою популярної бібліотеки **Swiper**, яка дозволяє легко створювати адаптивні та гнучкі каруселі.

У цьому випадку було підключено модуль **Navigation**, щоб додати кастомні стрілки навігації. Замість стандартних кнопок використовуються SVG-іконки SlideArrow, які стилізовані через модульні SCSS-класи. Для додаткового контролю над виглядом і позиціонуванням стрілок використовувалась бібліотека classNames для комбінації

класів: .swiper-button-prev / .swiper-button-next разом із власними стилями з Testimonials.module.scss.

Щоб створити ефект безперервної прокрутки, масив **testimonials** був розширений повторенням (використано три копії), завдяки чому карусель виглядає більш насиченою та не дає відчуття обмеженого контенту.
“const testimonialSlides = [...testimonials, ...testimonials, ...testimonials];”

Кожен слайд — це окремий компонент **TestimonialSlide**, який містить текст відгуку, фото клієнта, ім’я, роль (Customer) та візуальну оцінку у вигляді зірок. Весь слайдер обгорнутий у Container для дотримання загальної сітки сторінки та адаптивної структури.

4.6. Модальні вікна та взаємодія з користувачем

У сучасних веб-додатках, особливо в e-commerce-сегменті, модальні вікна відіграють важливу роль у комунікації з користувачем. Вони дозволяють оперативно передати повідомлення, запропонувати знижку, оформити замовлення або здійснити інші ключові дії — без потреби переходити на окрему сторінку. Таким чином, модальні вікна поєднують у собі зручність, швидкість доступу та гнучкість у відображенні динамічного контенту.

Функціональна роль у рамках інтернет-магазину

У межах розробленого інтерфейсу інтернет-магазину модальні вікна використовуються як візуальні тригери для залучення користувача до важливих дій:

- Підписка на розсилку;
- Швидкий перегляд товару;
- Додаткові повідомлення, як-от знижки, акції, оновлення.

Одним із найяскравіших прикладів такої взаємодії є модальне вікно підписки на новини — **NewsletterModal**, яке ми детально розглянемо нижче.

4.6.1 Вікно підписки на новини

NewsletterModal — це компонент, який реалізує спливаюче вікно з пропозицією підписки на новини та акції магазину. Його головне завдання — залучити користувача до участі в житті ресурсу, запропонувати цінність (наприклад, знижку), отримавши при цьому електронну адресу для подальшої комунікації.

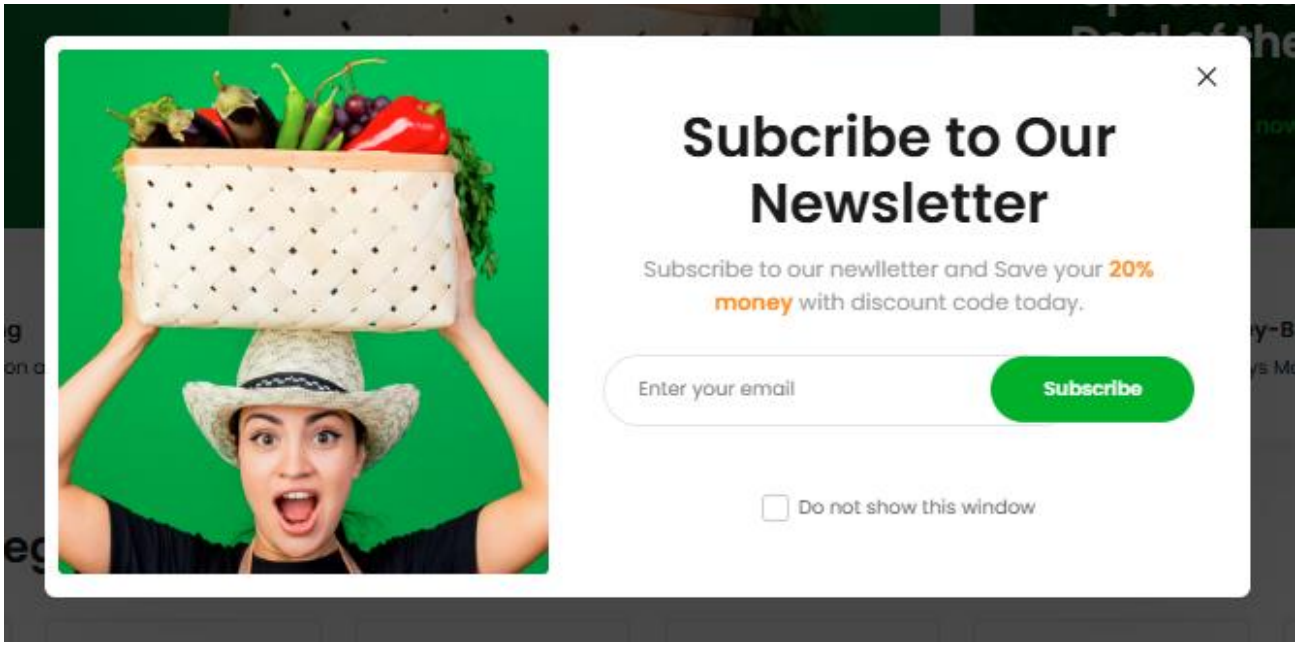

Рисунок 4.19 – Модальне вікно Newsletter

Компонент виконує наступні функції:

- автоматично з’являється через 5 секунд після завантаження сторінки;
- блокує фоновий контент до моменту закриття або взаємодії;
- дозволяє ввести електронну адресу для підписки;
- містить прапорець “Do not show this window”, який дозволяє уникнути повторної появи модального вікна (хоча сама реалізація цієї логіки не включена в межах цього компонента — вона передбачає збереження стану, наприклад, у localStorage);
- кнопка закриття з SVG-іконкою (**Close**) для ручного закриття вікна;
- візуально складається з двох основних частин: зображення товарів зліва та форми підписки справа.

Механіка роботи

Компонент побудований на основі React-хука useModal, який забезпечує логіку відкриття та закриття вікна. Додатково використовується хук useEffect, в якому створюється таймер на 5 секунд після першого завантаження сторінки:

```
useEffect(() => {
  if (!isOpen) {
    const timer = setTimeout(() => {
      openModal();
    }, 5000);

    return () => clearTimeout(timer);
  }
}, []);
```

Цей фрагмент коду забезпечує запуск модального вікна рівно через 5 секунд після завантаження сайту, але тільки якщо воно ще не було відкритим. Таймер очищається після демонтажу компонента, що відповідає правилам безпеки в React.

Для прапорця використовується useState<boolean>:

```
const [isChecked, setIsChecked] = useState(false);
```

Прапорець наразі лише змінює візуальний стан, але закладена можливість розширення логіки, наприклад, додавання збереження стану у локальне сховище, що дозволить не показувати вікно повторно тому ж користувачеві.

Зручність використання і повторне застосування

Компонент реалізований як автономна одиниця. Він не залежить від зовнішніх станів і може бути без змін повторно використаний у будь-якому іншому проєкті. Стилі компонента винесені в окремий SCSS-модуль **NewsletterModal.module.scss**, що гарантує відсутність конфліктів з іншими стилями на сайті.

Уся структура огорнута у спільний компонент **Modal**, який використовується і для інших типів вікон (наприклад, **QuickView**). Це забезпечує єдину архітектурну логіку модальних елементів усього додатку.

Кастомний хук useModal

Щоб забезпечити гнучке та повторно використовуване управління станом модальних вікон, було створено кастомний хук useModal. Його задача — абстрагувати логіку відкриття, закриття та перемикання стану модального вікна у зручну функцію, яку можна підключати до будь-якого компонента без дублювання коду.

Ось повна реалізація хука:

```
const useModal = () => {
    const [isOpen, setIsOpen] = useState<boolean>(false);

    const openModal = () => setIsOpen(true);
    const closeModal = () => setIsOpen(false);
    const toggleModal = () => setIsOpen(prev => !prev);

    return {
        isOpen,
        openModal,
        closeModal,
        toggleModal
    };
};
export default useModal;
```

Рисунок 4.20 – Кастомний хук useModal

Призначення та переваги:

- **Локалізований стан:** хук створює локальний булевий стан isOpen, який зберігає інформацію про те, чи відкрито модальне вікно.
- **Три методи управління:**
  - openModal() — відкриває модальне вікно;
  - closeModal() — закриває його;
  - toggleModal() — перемикає стан на протилежний (зручно при відкритті/закритті за одним тригером).
- **Універсальність**: підходить для будь-якого модального вікна на сайті — як для NewsletterModal, так і для QuickViewModal, CartModal, тощо.

- **Простота підключення:** достатньо викликати const { isOpen, openModal, closeModal } = useModal(); у компоненті, після чого можна безпосередньо передати значення та методи до Modal.

Таким чином, хук **useModal** реалізує шаблон інкапсуляції стану — одну з ключових практик при розробці компонентів у React. Він не лише скорочує код, але й дозволяє централізовано контролювати поведінку модальних вікон, підвищуючи зручність і надійність розробки.

4.6.2 Вікно швидкого перегляду

Модальне вікно швидкого перегляду (**QuickViewModal**) є ключовим елементом взаємодії користувача з каталогом товарів в інтернет-магазині. Його основна функція — надати можливість ознайомитися з деталями конкретного продукту без необхідності переходу на окрему сторінку. Такий підхід значно покращує зручність користування інтерфейсом, дозволяє заощадити час і спрощує процес прийняття рішення про покупку.

Відкриття модального вікна ініціюється при натисканні на спеціальну кнопку з іконкою у вигляді ока. Вона знаходиться біля кожного товару у списку, й виконує

функцію виклику методу onQuickView(product), передаючи до компонента дані про вибраний товар:

```
<button onClick={() => onQuickView(product)}>
    <Eye />
</button>
```

Компонент QuickViewModal побудований на основі загального компонента Modal, який відповідає за відображення напівпрозорого тла, центрованого вмісту та логіку закриття. Основна структура вікна розділена на дві частини: ліва зона — це слайдер зображень товару, права — детальна інформація та дії користувача.

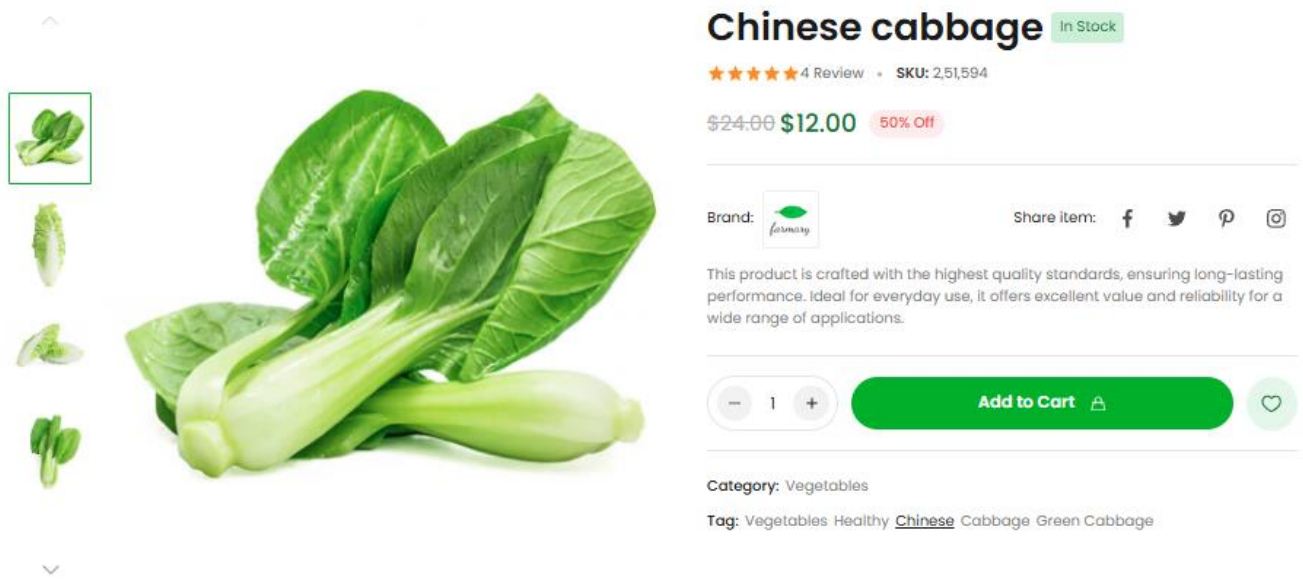

Рисунок 4.21 – Модальне вікно QuickView

Ліва колонка (галерея зображень)

Для реалізації галереї використано бібліотеку Swiper. Існує два взаємопов’язаних слайдери:

- **Thumbs Swiper** — вертикальний слайдер із мініатюрами, що дозволяє вибрати конкретне зображення.
- **Main Swiper** — головне зображення, яке змінюється залежно від вибраної мініатюри.

Стан активного слайда синхронізується через хуки **useState** та події кліку, які відслідковуються у **useEffect**. Усі зображення беруться з масиву productGallery[0].images:productGallery[0].images:

```
<Swiper
    onSwiper={setThumbsSwiper}
    direction={'vertical'}
    slidesPerView={4}
    spaceBetween={12}
    navigation={{
        nextEl: '.swiper-button-next',
        prevEl: '.swiper-button-prev'
    }}
    watchSlidesProgress
    modules={[Navigation, Thumbs]}
    className={styles.productThumbSlider}
>
    {productGallery[0].images.map((image, index) =>
        <SwiperSlide
            key={image.id}
            className={classNames(
                styles.productThumbSlide,
                { [styles.active]: index === activeIndex }
            )}
        >
            <div className={styles.productThumbImage}>
                <img
                    src={image.src}
                    alt={image.alt}
                />
            </div>
        </SwiperSlide>
    )}
</Swiper>
```

Рисунок 4.22 – Галерея зображень

Права колонка (інформація про товар)

Праворуч розміщується повна інформація про товар:

- Назва продукту — відображається у заголовку.
- Наявність — текстове повідомлення «In Stock».
- Рейтинг — кількість зірок та кількість відгуків (у прикладі: 4 Review).
- Артикул (SKU) — умовний унікальний ідентифікатор товару.
- Ціна — стара та поточна ціна, якщо є знижка.
- Відсоток знижки — обчислюється за допомогою функції **calculateDiscount**:

  *export const **calculateDiscount** = (product: ProductItem): string => {*

  if (!product.isSale || !product.salePrice) {

  return '0%'; // If there's no sale price, return 0 (no discount)

  } else {

  *const discountPct = ((product.price - product.salePrice) / product.price) * 100;*

  return `${Math.round(discountPct)}%`;

  }

  };

  Функція перевіряє, чи товар дійсно продається за акційною ціною. Якщо ні — повертається значення '0%', тобто знижка відсутня. У випадку, коли товар має як звичайну, так і знижену ціну, функція обчислює різницю між ними, переводить її у відсотковий еквівалент і повертає результат у вигляді текстового значення з символом відсотка. Цей відсоток потім використовується для візуального відображення вигоди для користувача, наприклад, як позначка «50% Off» у модальному вікні швидкого перегляду товару.

Також вказується бренд (із зображенням логотипа), присутні іконки для поширення товару через соціальні мережі (<SocialLinks />) та короткий опис продукту, що дозволяє сформувати у користувача уявлення про якість і цінність товару.

Інтерактивні елементи управління

Під основною інформацією розміщується набір елементів взаємодії:

- Лічильник кількості — кнопки + та –, реалізовані через кастомний хук **useCounter**
- Кнопка “**Add to Cart**” — додає товар до кошика.
- Кнопка “**Wishlist**” — додає до списку бажаного.

Мета-дані товару

У нижній частині також містяться додаткові категорійні характеристики:

- **Категорія** (наприклад: Vegetables)
- **Теги** (наприклад: Healthy, Chinese, Cabbage тощо), які сприяють точнішій фільтрації та пошуку.

Завдяки реалізації модального вікна швидкого перегляду (QuickViewModal) користувач має змогу ознайомитися з ключовою інформацією про товар, не покидаючи поточної сторінки. Такий підхід покращує зручність використання сайту, зменшує кількість зайвих переходів та допомагає швидше ухвалювати рішення.

4.7 Сторінка Shop

Сторінка **Shop** є основною вітриною інтернет-магазину, що дозволяє користувачам переглядати весь асортимент товарів, фільтрувати їх за категоріями та переглядати знижкові пропозиції. Ця сторінка слугує для зручного вибору продуктів перед їх додаванням у кошик або переглядом детальної інформації.

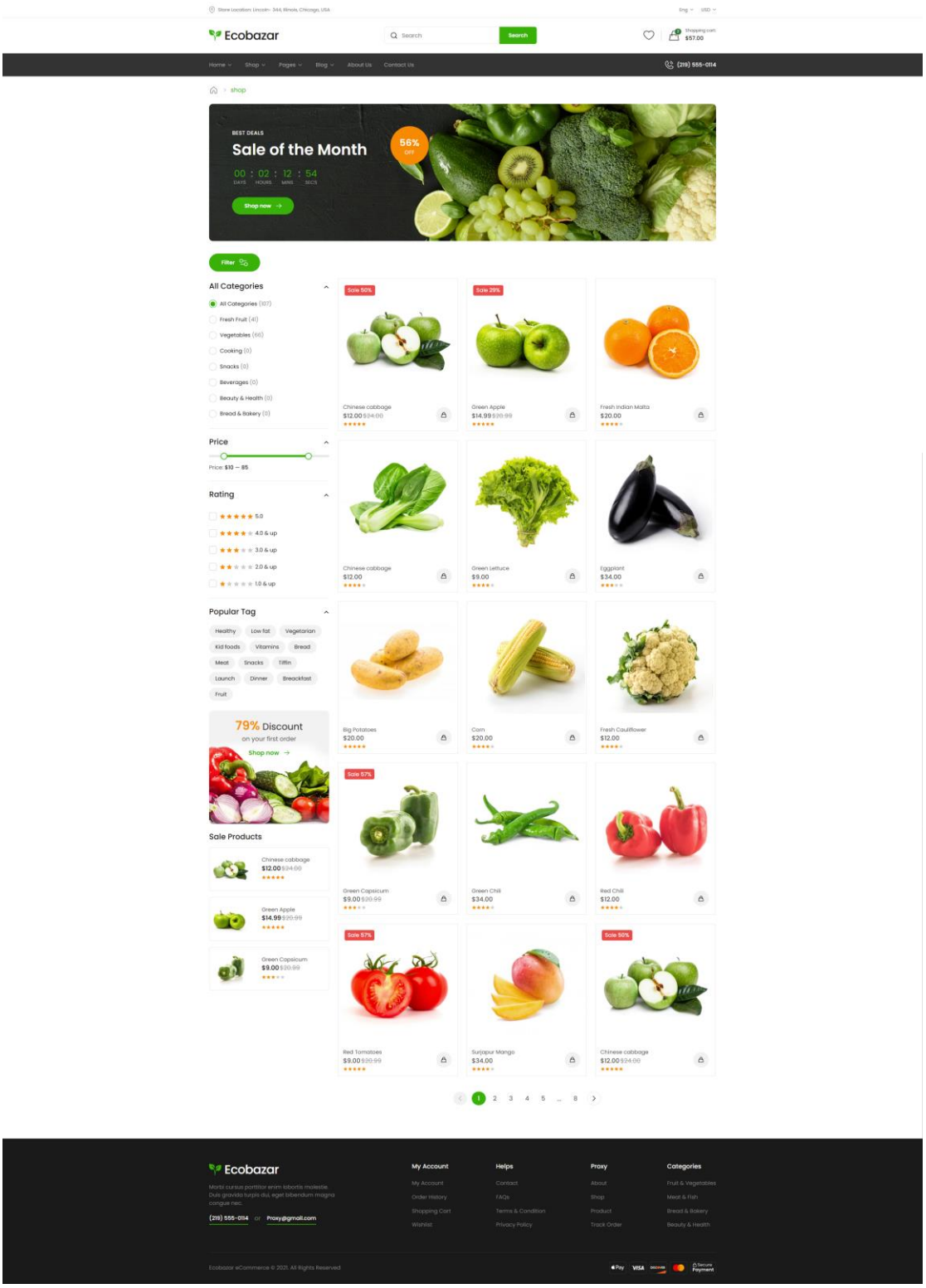

Рисунок 4.23 – Сторінка Shop

Доступ до сторінки

Перед переходом на сторінку Shop, користувач повинен **авторизуватись** або **зареєструватись**. Це реалізовано шляхом перевірки стану автентифікації

користувача у глобальному сховищі (Redux). Такий підхід дозволяє обмежити доступ до функціоналу магазину лише зареєстрованим користувачам.

Реєстрація

Під час реєстрації система виконує такі перевірки:

- коректність електронної пошти (перевірка формату);
- правильність пароля (відповідність вимогам безпеки);
- підтвердження пароля — паролі мають збігатися;
- прийняття умов використання (через чекбокс).

У разі успішної реєстрації користувач автоматично входить у систему та перенаправляється на головну сторінку

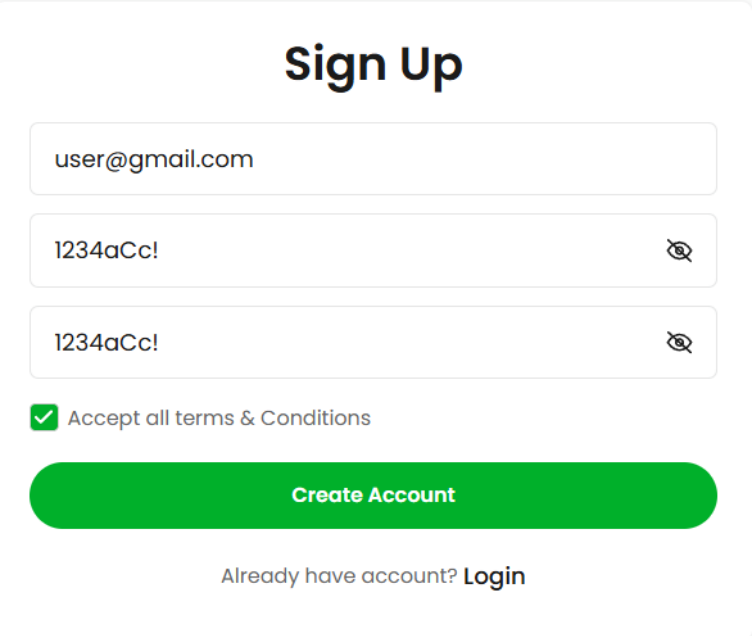

Рисунок 4.24 – Компонент Реєстрації

Авторизація

Форма входу реалізує:

- валідацію введених email та пароля;
- можливість обрати “Запам’ятати мене” (чекбокс);
- у разі успіху — перенаправлення на головну сторінку.

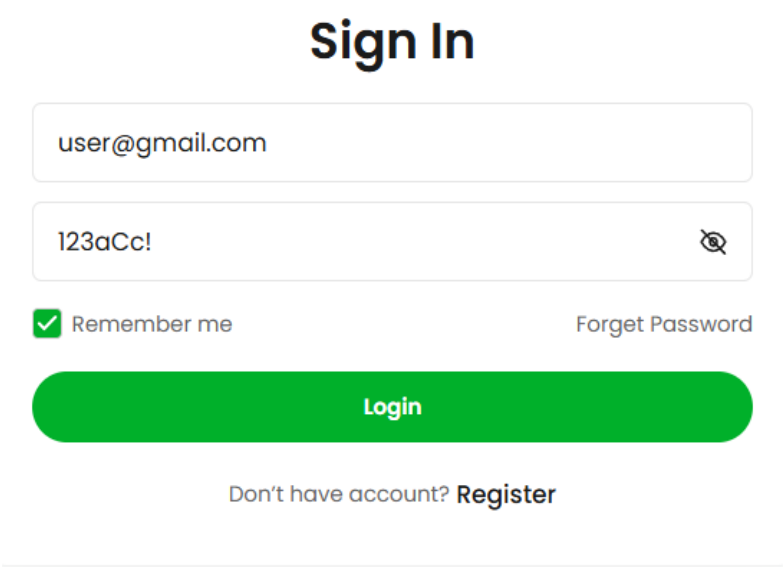

Рисунок 4.25 – Компонент Авторизації

На самій сторінці **Shop** реалізовано такі елементи:

- **Breadcrumb** — хлібні крихти, що відображають поточне положення користувача на сайті.
- **MonthlyBanner** — рекламний банер місяця зі спеціальною пропозицією.
- **Filter** — бічна панель із фільтрами, включаючи блок зі знижковими товарами.
- **Список товарів**, який динамічно генерується за допомогою компонента Product. Виводиться кожен товар із ключовою інформацією.
- **Pagination** — пагінація, яка дозволяє переходити між сторінками при великій кількості товарів.

Дані про товари надходять із масиву products, який імпортується зі статичних даних. Окремо відбираються товари зі знижками (saleProducts), які передаються у фільтр.

Сторінка Shop є центральним елементом сайту, де реалізовані всі необхідні функції для взаємодії з товарами.

Логіка відображення та розбиття на сторінки

Для покращення користувацького досвіду реалізована пагінація — механізм, що дозволяє не виводити одразу всі товари, а ділити їх на сторінки. Це досягається за допомогою компонента Pagination, який приймає такі параметри:

- **totalPages** — загальна кількість сторінок;
- **currentPage** — поточна сторінка;
- **setPage** — функція для зміни активної сторінки.

На початку він генерує масив сторінок від 1 до **totalPages**:

```
const pages: number[] = [];
for (let i = 0; i < totalPages; i++) {
    pages.push(i + 1);
}
```

Далі визначається, чи поточна сторінка є першою або останньою, що потрібно для блокування навігаційних кнопок, якщо подальша навігація неможлива:

```
const isFirstPage = currentPage === pages[0];
const isLastPage = currentPage === pages[pages.length - 1];
```

Також при першому рендері сторінки через **useEffect** викликається обчислення загальної кількості сторінок (**pageCount**) на основі кількості товарів і ліміту, після чого результат записується у глобальний стан (**dispatch(setTotalCount(pageCount))**).

Логіка кнопок навігації

Компонент має дві стрілки: назад (вліво) та вперед (вправо). При натисканні на стрілки викликається відповідна функція:

```
const handlePrevClick = () => {
    if (totalPages > 7 && isLastPage) {
        setPage(pages[4]);
```

```
        } else {
            setPage(currentPage - 1);
        }
}
const handleNextClick = () => {
        if (totalPages > 7 && currentPage === pages[4]) {
            setPage(pages[pages.length - 1]);
        } else {
            setPage(currentPage + 1);
        }
}
```

Ця логіка дозволяє додатково керувати переміщенням при великій кількості сторінок: наприклад, швидко переходити до середини або кінця списку сторінок.

Особливості компонента пагінації

Компонент **Pagination** розраховує кількість сторінок динамічно та виводить список із кнопками. Якщо сторінок більше семи, після п’ятої відображається елемент з трьома крапками («...») для візуального скорочення довгого списку. Кнопки навігації «вперед» та «назад» автоматично відключаються на першій і останній сторінках відповідно.

Кнопка зліва виконує перехід на попередню сторінку, праворуч — на наступну. Для зручності реалізоване спеціальне перемикання в межах великого списку сторінок, щоб дозволити користувачу швидко повернутись до початку або перейти в кінець.

Інтерфейс фільтрації товарів (Filter)

На сторінці Shop зліва розміщується панель фільтрації, яка дозволяє користувачеві звузити вибір товарів за різними критеріями. Її мета — надати швидкий та зручний спосіб знайти потрібні продукти без потреби переглядати всю товарну лінійку.

Фільтри інтерактивні, і при зміні параметрів автоматично оновлюється список відображуваних товарів.

До основних елементів блоку **Filter** входять:

- **Кнопка** “**Filter**” — відкриває або згортає панель фільтрації, що зручно для мобільних пристроїв або невеликих екранів.
- **Категорії товарів (All Categories)** — список із можливістю обрати одну з категорій (наприклад, Fresh Fruits, Vegetables, Snacks тощо). Кожна категорія супроводжується кількістю доступних товарів у дужках.
- **Фільтр за ціною (Price)** — реалізовано за допомогою повзунка, який дозволяє встановити діапазон цін. Значення діапазону динамічно змінюються у відповідності до мінімальної та максимальної вартості товарів у базі.
- **Рейтинг (Rating)** — представлений у вигляді зірочок (від 1 до 5). Користувач може обрати, наприклад, тільки товари з рейтингом 4.0+ або 5.0.
- **Популярні теги (Popular Tag)** — набір ключових тегів, які відображають тематику або властивості продуктів (наприклад: Healthy, Vegan, Vitamin, Lunch, Fruit). Обрання тега застосовує відповідний фільтр.
- **Рекламний банер** — окремий блок з промоакцією, який акцентує увагу на знижці при першому замовленні (наприклад, "79% Discount").
- **Товари зі знижками (Sale Products)** — динамічний список декількох акційних товарів з візуалізацією зниженої ціни, початкової ціни та рейтингу.

Ці елементи у поєднанні дозволяють користувачу швидко налаштувати відображення каталогу відповідно до його запитів. Це особливо корисно при великій кількості товарів і підвищує зручність користування сайтом.

ВИСНОВОК

У межах дипломної роботи було реалізовано повноцінну фронтенд-частину веб-застосунку для інтернет-магазину з урахуванням сучасних підходів до розробки, організації коду та його подальшої підтримки. Основна мета полягала у створенні зручного та зрозумілого інтерфейсу, який легко масштабувати та доповнювати новим функціоналом у майбутньому.

Початковий етап роботи включав налаштування проєктного середовища: сформовано структуру папок і файлів, налаштовано **Craco** для зручного використання абсолютних шляхів імпорту (наприклад, @/, @images). Це значно спростило навігацію між модулями та зменшило кількість помилок під час імпорту компонентів, стилів або ресурсів.

Фронтенд частина реалізована на базі бібліотеки **React** з використанням **TypeScript**, що дало можливість краще контролювати типізацію даних та уникнути багатьох помилок на етапі розробки. Стилі побудовані з використанням **SCSS**, що дозволило створити змінні для кольорів, відступів та інших повторюваних значень, а також забезпечити підтримку єдиного стилю по всьому проєкту. Базові стилі (наприклад, для кнопок, карток товарів, бейджів, таймерів) були винесені в окремі файли для повторного використання та легшого супроводу.

Реалізовано низку ключових компонентів: великі й малі картки товарів, кнопки, рейтинг, категорії, банери, таймери акцій, а також базові елементи структури сторінки, такі як шапка (header), навігація тощо. З метою покращення читабельності та зручності підтримки, компоненти поділені на модулі — наприклад, **Product/BigProduct**, **Product/SmallProduct**, кожен з яких має окремі стилі, але при цьому використовує спільні базові класи, винесені в Product.module.scss.

Для керування глобальним станом застосунку інтегровано **Redux Toolkit**. Це дало змогу централізовано працювати з кошиком, даними користувача та іншими

важливими станами. Реалізовано просту форму авторизації: при введенні даних стан автоматично оновлюється відповідно до дій користувача.

Окрема сторінка проєкту демонструє головний функціонал магазину: банери, категорії, картки товарів із рейтингом, кнопки “Купити” та інші елементи, що дозволяють оцінити загальний вигляд інтерфейсу. Дані про товари, категорії, авторів і коментарі були винесені в окремі файли та типізовані, що спростило процес їх оновлення й підтримки, а також покращило загальну організацію проєкту.

Підсумком роботи стало створення адаптивного веб-інтерфейсу інтернет-магазину, який можна розвивати далі, доповнюючи новими сторінками, функціями та інтегруючи з бекендом. Реалізований застосунок демонструє впевнене володіння інструментами React, TypeScript, Redux Toolkit, SCSS та сучасними принципами фронтенд-розробки. Такий підхід може стати основою для подальшої розробки повнофункціонального онлайн-магазину, придатного до використання в реальних умовах.

СПИСОК ЛІТЕРАТУРИ

1. React – офіційна документація [Електронний ресурс]. – Режим доступу: https://react.dev – Назва з екрана.
2. Redux – офіційна документація [Електронний ресурс]. – Режим доступу: https://redux.js.org – Назва з екрана.
3. JavaScript – офіційна документація Mozilla Developer Network (MDN) [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/ru/docs/Web/JavaScript – Назва з екрана.
4. HTML5 – офіційна документація MDN [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/ru/docs/Web/Guide/HTML/HTML5 – Назва з екрана.
5. CSS/SCSS – документація Sass [Електронний ресурс]. – Режим доступу: https://sass-lang.com/guide – Назва з екрана.
6. Node.js – офіційна документація [Електронний ресурс]. – Режим доступу: https://nodejs.org/en/docs – Назва з екрана.
7. npm (Node Package Manager) – документація [Електронний ресурс]. – Режим доступу: https://docs.npmjs.com – Назва з екрана.
8. React Router – документація [Електронний ресурс]. – Режим доступу: https://reactrouter.com – Назва з екрана.
9. Create React App – офіційна документація [Електронний ресурс]. – Режим доступу: https://create-react-app.dev – Назва з екрана.
10. Git та GitHub – офіційна документація [Електронний ресурс]. – Режим доступу: https://git-scm.com/doc – Назва з екрана.
11. ESLint – документація з налаштування лінтингу коду [Електронний ресурс]. – Режим доступу: https://eslint.org/docs/latest/ – Назва з екрана.
12. Prettier – документація по автоформатуванню коду [Електронний ресурс]. – Режим доступу: https://prettier.io/docs/en/index.html – Назва з екрана.

13. Babel – офіційна документація [Електронний ресурс]. – Режим доступу: https://babeljs.io/docs/en/ – Назва з екрана.
14. Webpack – документація по збірці проєктів [Електронний ресурс]. – Режим доступу: https://webpack.js.org/concepts/ – Назва з екрана.

ДОДАТОК А. Код програми

Повний код веб-додатку доступний у репозиторії за посиланням:
https://github.com/Devf1s/ecobazar

ДОДАТОК Б. Перелік тестових питань

1. Яка основна перевага SPA-додатка порівняно з класичними веб-сайтами?
    - Швидша взаємодія без перезавантаження сторінки
    - Потреба в більшій кількості серверів
    - Залежність від мобільних додатків
    - Використання XML-структур

2. Який з наведених фреймворків було використано для створення інтерфейсу веб-додатку?
    - Angular
    - Vue.js
    - React
    - Ember.js

3. Для чого використовується Redux у проєкті?
    - Для створення візуальних компонентів
    - Для навігації між сторінками
    - Для керування глобальним станом додатку
    - Для взаємодії з базою даних

4. Яким чином реалізовано захист сторінки магазину від неавторизованих користувачів?
    - Через localStorage
    - Через обмеження маршрутизатора (Router Guard)
    - Через cookies
    - Через окремий компонент стилізації

5. Яка роль компонента useEffect у React?

- Виводить стилі на сторінку
- Дозволяє виконувати побічні ефекти, наприклад, запити до API
- Створює маршрути
- Додає події кліку

6. Який метод HTTP використовується для реєстрації нового користувача?

- GET
- PUT
- POST
- DELETE

7. Що відбувається при успішній авторизації користувача?

- Користувач перенаправляється на головну сторінку магазину
- Сторінка перезавантажується
- Пароль зберігається у відкритому вигляді
- Браузер закривається

8. Яка бібліотека використовується для маршрутизації сторінок у додатку?

- redux-thunk
- react-router-dom
- express
- bootstrap

9. Що таке "пагінація" у контексті веб-додатку?

- Механізм для перевірки паролів
- Метод сортування за алфавітом
- Спосіб розбиття контенту на сторінки
- Процес стиснення зображень

10. Що таке компонент Product у додатку?

- Сторінка оплати
- Компонент, що відображає товар
- Сторінка логіна
- API-запит

11. Як реалізовано фільтрацію товарів у додатку?

- Через окрему базу даних
- Через параметри URL-запиту
- Через стилі CSS
- Через cookies

12. Яке сховище найчастіше використовується для збереження токена авторизації?

- Redux Store
- sessionStorage
- localStorage
- IndexedDB

13. Який формат даних передається між frontend і backend у додатку?

- XML
- JSON
- CSV
- YAML

14. Який елемент дозволяє користувачу додавати товар до кошика?

- Компонент Navbar
- Кнопка "Add to cart"
- Кнопка "Logout"

- Footer

15. Яка функція використовується для надсилання HTTP-запитів у проєкті?

- alert()
- useParams()
- fetch() або axios()
- console.log()

16. Що означає стан isLoading у Redux або React?

- Дані не існують
- Помилка під час авторизації
- Зараз відбувається завантаження даних
- Користувач вийшов з системи

17. Яка роль dispatch у Redux?

- Створює компоненти
- Змінює DOM напряму
- Відправляє дію (action) до стору
- Зберігає HTML-розмітку

18. Який хук React дозволяє створювати та оновлювати стан компоненту?

- useRoute
- useEvent
- useState
- useMemo

19. Який з нижченаведених є правильним шляхом до динамічного маршруту?

- /product
- /product/:id
- /:product

- /product=id

20. Який з цих елементів є прикладом умовного рендерингу в React?

- if (true) return JSX;
- console.log("Render")
- import React from 'react'
- return <Router />