МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії

"До захисту допущена"
Зав. кафедри комп'ютерних наук та прикладної математики Турбал Ю.В.
_____
«___» _____ 20__ р.

КВАЛІФІКАЦІЙНА РОБОТА

Проєктування та розробка веб-додатку для магазину гаджетів на основі технології React

Виконав: **Войтович Андрій Вікторович** _____
(прізвище, ім'я, по-батькові) (підпис)

група КН-41

Керівник: **доцент Демчук О.С.** _____
(науковий ступінь, вчене звання, посада, прізвище, ініціали) (підпис)

Рівне-2024

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота: 64 с., 36 малюнків, 9 джерел, 2 додатки.

Мета кваліфікаційної роботи: розробити та впровадити сучасний веб-додаток для електронної комерції, який забезпечить зручний та інтуїтивно зрозумілий інтерфейс для користувачів.

Об’єкт дослідження: процес розробки веб-додатків для електронної комерції.

Предмет дослідження: технології та методи, що використовуються для створення фронтенд та бекенд частин веб-додатку, включаючи React, Redux, Node.js та Express.js.

Методи дослідження: аналіз літератури, проєктування архітектури додатку, програмування, тестування, а також використання інструментів для забезпечення продуктивності та безпеки веб-додатку.

У зв’язку із тим, що розробка ефективних веб-додатків для електронної комерції є критично важливою для бізнесів, реалізований у даній роботі веб-застосунок усуває основні проблеми сучасних рішень – низьку продуктивність та складність користувацького інтерфейсу.

Ключові слова: ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ, ВЕБ-ДОДАТОК, ВЕБ-ЗАСТОСУНОК, ЕЛЕКТРОННА КОМЕРЦІЯ, JAVASCRIPT, NODEJS, REACT, TYPESCRIPT, БАЗА ДАНИХ, CSS, SASS.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

IT – Information Technology

NPM – Node Package Manager

SPA – Single Page Application

DOM – Document Object Model

CSS – Cascading Style Sheets

SASS – Syntactically Awesome Style Sheets

FSD – Feature Sliced Design

MVC – Model-View-Controller

URL – Uniform Resource Locator

ВСТУП

У сучасному світі інформаційні технології відіграють ключову роль у розвитку різних галузей, включаючи електронну комерцію. З кожним роком зростає кількість онлайн-магазинів, що пропонують користувачам зручний та швидкий доступ до товарів і послуг. Водночас, зростають і вимоги до якості веб-додатків, що використовуються для реалізації таких магазинів.

Інтернет-магазини стали невід'ємною частиною нашого життя, пропонуючи споживачам широкий вибір товарів, зручність покупок у будь-який час доби та можливість порівняти ціни та відгуки. Згідно з останніми дослідженнями, обсяг світового ринку електронної комерції щорічно збільшується, і очікується, що цей тренд зберігатиметься в найближчі роки. Однією з ключових складових успішного інтернет-магазину є його веб-додаток, який повинен бути швидким, надійним, зручним у використанні та привабливим для користувачів.

Конкуренція на ринку інтернет-магазинів постійно зростає, що спонукає розробників до впровадження інноваційних рішень і покращення функціоналу своїх платформ. В умовах постійно змінюваного ринку і швидкого розвитку технологій, підприємства, які своєчасно адаптуються до нових умов і використовують передові технології, такі як React, мають значні конкурентні переваги. Впровадження сучасних технологій дозволяє підвищити ефективність бізнес-процесів, покращити взаємодію з клієнтами та збільшити рівень продажів.

Метою даної роботи є проєктування та розробка веб-додатку для магазину гаджетів, що забезпечить зручний, швидкий і надійний інтерфейс для користувачів. Для досягнення цієї мети необхідно виконати такі завдання:

- провести аналіз існуючих рішень для інтернет-магазинів та визначити їх переваги і недоліки.

- обґрунтувати вибір технологій, що будуть використані для розробки веб-додатку.
- спроєктувати архітектуру системи та розробити API.
- реалізувати інтерфейс користувача та основний функціонал веб-додатку.
- провести тестування і налагодження додатку.

Об’єктом дослідження даної роботи є технологія React, використання технологій JavaScript у розробці веб-додатків для інтернет-магазинів, наявні підходи використання React у сфері електронної комерції.

РОЗДІЛ 1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Застосування інформаційних технологій для онлайн комерції

Одним досить актуальним явищем в Україні зараз є діджиталізація, яка після соціальної сфери проникла і в комерційну діяльність. З кожним роком все більше покупок здійснюється через Інтернет, що змушує підприємців у сфері торгівлі та послуг задуматися про залучення додаткових споживачів з онлайн-простору або навіть про повну цифрову трансформацію свого бізнесу, що було особливо важливо в період пандемії коронавірусу. Проте, існує багато варіантів діджиталізації торгівлі, тому продавці стикаються з проблемою вибору формату інтернет-торгівлі та технологій для його впровадження.

Інформаційні технології (ІТ) є рушійною силою для сучасної онлайн-комерції, забезпечуючи платформу для взаємодії покупців і продавців у цифровому середовищі. Використання ІТ у цій сфері значно розширює можливості бізнесу, підвищує ефективність операцій та покращує клієнтський досвід.

Інтернет-магазини можуть впроваджувати різноманітні інструменти аналітики для відстеження поведінки покупців та оптимізації маркетингових стратегій.

Завдяки ІТ-технологіям, бізнеси отримують можливість автоматизувати управління запасами, інтегрувати зручні платіжні системи та забезпечувати високий рівень безпеки даних. Крім того, технології штучного інтелекту та машинного навчання дозволяють створювати персоналізовані рекомендації для користувачів, що сприяє підвищенню продажів та лояльності клієнтів.

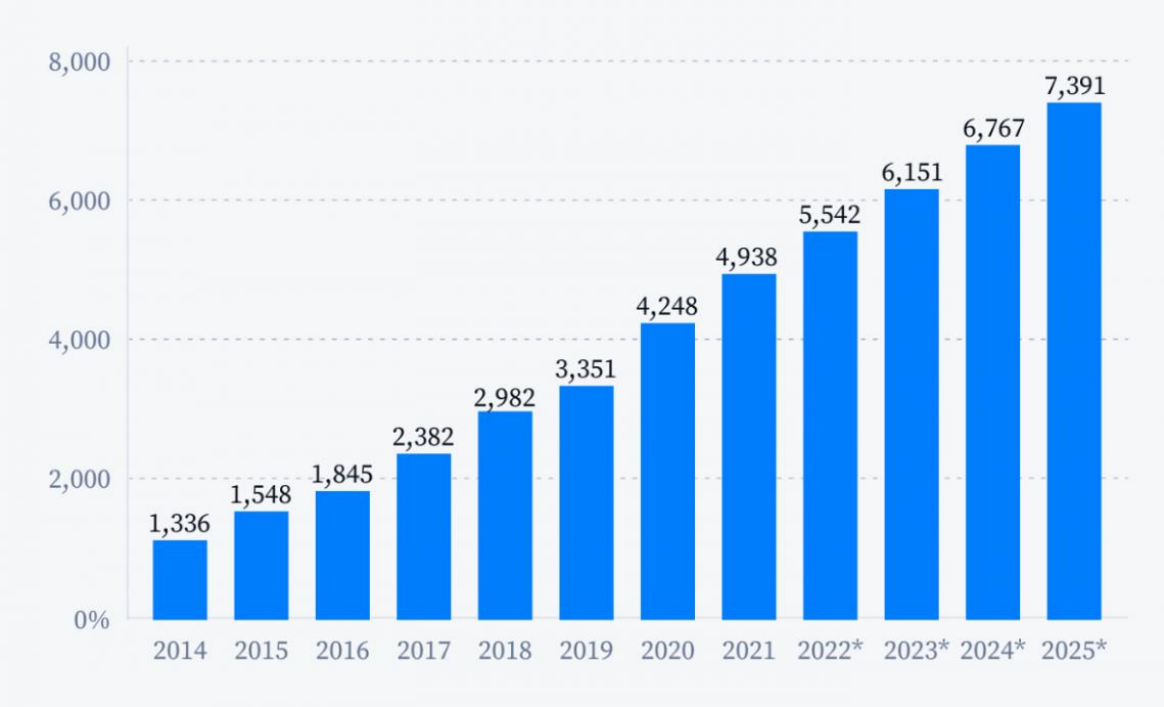


Рис. 1.1. Продажі електронної комерції по всьому світу в млрд. USD

1.2. Типи інтернет магазинів

Перед початком створення власного інтернет-магазину, важливо ознайомитися з різними видами та категоріями таких платформ, щоб вибрати найбільш відповідну для ваших вимог і цілей. Розглянемо основні типи інтернет-магазинів та їхні характеристики.

B2C (Business-to-Consumer)

Цей формат інтернет-магазину призначений для продажу товарів або послуг безпосередньо кінцевим споживачам, тобто фізичним особам. B2C магазини є найпоширенішими серед онлайн-роздрібників і пропонують широкий спектр продуктів, включаючи одяг, взуття, електроніку та косметику. Приклади B2C магазинів: Amazon, eBay, Zappos. Ця модель бізнесу стала дуже популярною завдяки своїй зручності та легкому доступу для клієнтів. Такі магазини надають великий вибір товарів і послуг, а також можуть використовувати SEO просування для підвищення видимості та конкурентоспроможності в мережі.

B2B (Business-to-Business)

Цей тип інтернет-магазину обслуговує інші компанії, постачаючи їм товари або послуги. Зазвичай такі магазини пропонують продукцію оптом або надають спеціалізовані послуги для бізнесу. Приклади B2B магазинів: Alibaba, Office Depot, Grainger.

C2C (Consumer-to-Consumer)

Дані платформи дозволяють користувачам продавати товари або послуги одне одному. Головною метою таких платформ є створення середовища для обміну товарами між фізичними особами. Найвідоміші C2C платформи: eBay, Etsy, Craigslist.

P2P (Peer-to-Peer)

Такі платформи дозволяють користувачам надавати послуги одне одному. Це може включати оренду житла, автомобілів або обладнання, а також надання різноманітних послуг. Приклади P2P платформ: Airbnb, Turo, TaskRabbit.

Оптові та Роздрібні Магазини

Оптові магазини зосереджені на продажу товарів великими партіями, зазвичай для інших бізнесів або посередників. Роздрібні магазини пропонують товари безпосередньо кінцевим споживачам. Багато інтернет-магазинів поєднують обидві моделі, надаючи можливість купувати як оптом, так і вроздріб.

Маркетплейс

Маркетплейс — це інтернет-платформа, де різні продавці можуть пропонувати свої товари або послуги. Такі платформи об'єднують широкий асортимент продукції від різноманітних постачальників на одному сайті. Прикладами популярних маркетплейсів є Amazon, eBay та Etsy.

Вибір типу інтернет-магазину має базуватися на специфіці певного бізнесу, потребах цільової аудиторії та конкурентній ситуації на ринку.

Правильний вибір допоможе ефективно використовувати можливості Інтернету для розвитку бізнесу і досягнення успіху в онлайн-торгівлі.

1.3. Підходи до побудови інтернет-магазинів

Важливо враховувати технічні аспекти при створенні та управлінні інтернет-магазином. Потрібно вибрати надійну та масштабовану платформу для створення вашого магазину, яка відповідає потребам вашого бізнесу.

Побудова інтернет-магазинів може здійснюватися за різними підходами, кожен з яких має свої особливості, переваги та недоліки. Одним з найпоширеніших підходів є використання готових платформ для електронної комерції, таких як Shopify, WooCommerce або Magento. Ці платформи надають широкий спектр інструментів для створення та управління онлайн-магазинами, що дозволяє швидко запустити проєкт з мінімальними технічними знаннями.

Основні переваги цього підходу включають швидкість розгортання, доступність технічної підтримки та регулярні оновлення. Проте, такі платформи можуть бути обмеженими у функціональності та дизайні, що може вплинути на здатність бізнесу адаптуватися до специфічних потреб ринку.

Іншим підходом є розробка індивідуальних рішень на основі фреймворків, таких як Vue, Angular або бібліотеки React, які дозволяють створювати більш гнучкі та налаштовані під потреби бізнесу інтернет-магазини. Цей підхід дає можливість реалізувати унікальні функції та інтеграції, які неможливо досягти за допомогою готових платформ. Основні переваги включають високу гнучкість, масштабованість та контроль над усіма аспектами магазину. Однак, даний підхід вимагає значних інвестицій у розробку та підтримку, а також високого рівня технічних знань.

Вибір підходу до побудови інтернет-магазину залежить від багатьох факторів, включаючи бюджет, технічні можливості команди та специфічні

потреби бізнесу. Інтеграція з різними платіжними системами, зручний інтерфейс користувача та ефективні маркетингові інструменти є ключовими елементами успішного інтернет-магазину.

1.4. Огляд та аналіз популярних інтернет-магазинів

ROZETKA є одним з найбільших і найпопулярніших інтернет-магазинів в Україні, що пропонує широкий асортимент товарів, від електроніки до побутової техніки та товарів для дому.

Веб-застосунок ROZETKA відзначається зручним інтерфейсом та високою швидкістю роботи, що забезпечує позитивний досвід для користувачів. Основний функціонал включає пошук товарів, фільтрацію за різними параметрами, додавання товарів у кошик та оформлення замовлення. Крім того, ROZETKA пропонує користувачам можливість залишати відгуки, порівнювати товари та використовувати програму лояльності.

Переваги веб-додатку включають великий асортимент товарів, швидку та зручну навігацію. Проте, іноді користувачі стикаються з технічними збоями під час високого навантаження, наприклад, у період розпродажів, що може впливати на зручність використання.

OLX представляє собою онлайн-платформу для купівлі-продажу товарів і послуг між приватними особами, яка стала надзвичайно популярною в Україні. Веб-застосунок OLX має простий та інтуїтивно зрозумілий інтерфейс, що дозволяє користувачам легко створювати оголошення, переглядати пропозиції та контактувати з продавцями.

Основний функціонал включає можливість розміщення безкоштовних оголошень, використання фільтрів для пошуку, а також захищені угоди за допомогою платформи OLX Delivery. Переваги веб-додатку включають велику

кількість користувачів, що забезпечує широкий вибір товарів, та можливість прямого контакту з продавцями.

Недоліками є ризик шахрайства через відсутність повного контролю над продавцями та обмежена відповідальність платформи за якість товарів, що продаються, а також інколи некоректна робота фільтрів пошуку.

АЛЛО ще один великий гравець на українському ринку електронної комерції, спеціалізується на продажу електроніки, гаджетів та побутової техніки.Веб-застосунок АЛЛО відзначається сучасним дизайном та зручністю у використанні, що дозволяє швидко знаходити потрібні товари, додавати їх у кошик та оформлювати замовлення.

Основний функціонал включає розширені фільтри пошуку, порівняння товарів, відгуки покупців та спеціальні пропозиції для зареєстрованих користувачів. АЛЛО також надає можливість скористатися послугами кредитування та розстрочки, що є значною перевагою для багатьох покупців.

Переваги веб-додатку АЛЛО включають зручний інтерфейс, можливість порівняння товарів та наявність програм лояльності.

Недоліки можуть включати іноді довгі терміни доставки та обмежений асортимент у порівнянні з конкурентами, що може впливати на вибір користувачів.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБ-ДОДАТКУ

2.1. Вибір стеку технологій

JavaScript є однією з найпоширеніших мов програмування, що використовується для розробки веб-додатків. Вона була створена для того, щоб надавати можливість динамічної взаємодії з користувачем на веб-сторінках, і з тих пір значно розширила свої можливості.

JavaScript є мовою високого рівня, що означає, що вона має синтаксис, близький до людської мови, що полегшує її вивчення і використання.

Переваги:

- JavaScript можна використовувати для створення як клієнтських, так і серверних додатків, що робить його дуже універсальною мовою.
- Широкий вибір фреймворків і бібліотек для зручної розробки.
- Підтримка асинхронного програмування, що робить його ефективним для обробки багатьох запитів одночасно без блокування виконання інших завдань.
- Велика та активна спільнота розробників, яка створює багато корисних інструментів.

Недоліки:

- Є різниця між реалізаціями JavaScript у різних браузерах, що може призводити до проблем з сумісністю.
- JavaScript є слабо типізованою мовою, що може призводити до помилок через невідповідність типів під час виконання програми.
- В деяких сценаріях JavaScript може бути менш продуктивним за інші мови, особливо в обчислювально інтенсивних задачах.

2.1.1. Typescript

TypeScript є суперсетом JavaScript, який вирішує один з найвагоміших його недоліків — додає строгу типізацію.

Основними перевагами TypeScript є здатність визначати типи даних для змінних, параметрів функцій, об'єктів і інших конструкцій коду. Це дозволяє виявляти помилки ще на етапі написання коду, що полегшує відлагодження та підтримку додатків у подальшому.

TypeScript дозволяє підвищити надійність програмного забезпечення, зменшити кількість помилок і покращити читабельність коду шляхом використання типових анотацій. Він допомагає уникнути ситуацій, коли функції отримують неправильні типи даних або коли змінні використовуються неочікуваним чином. Це забезпечує більшу стабільність програми і дозволяє розробникам зосередитися на логіці додатку, а не на вирішенні найтиповіших помилок.

Оскільки TypeScript є частиною сучасного екосистеми JavaScript, підтримується інтеграцією з популярними фреймворками та бібліотеками, такими як React, Angular і Vue.js. Використання TypeScript з цими інструментами сприяє покращенню інструментів розробки, автодоповнення коду, підказок і перевірок типів під час написання коду.

```
export type Product = {
  id: string;
  namespaceId: string;

  name: string;
  capacity: string;
  capacityAvailable: string[];
  priceRegular: number;
  priceDiscount: number;

  image: string;
  images?: Image[];
  colors: Color[];
  description: Description[];

  screen: string;
  resolution: string;
  processor: string;
  ram: string;
  camera: string;
  zoom: string;
  cell: string[];

  discount?: number;
  category: { name: string }

  time_created: string;
  time_updated: string;
}
```

Рис. 2.1. Приклад типізації продукту

2.1.2. Node.js

Node.js є середовищем виконання JavaScript, побудованим на двигуні V8, який також використовується у браузері Google Chrome.

Основною особливістю Node.js є те, що він дозволяє виконувати JavaScript на серверному боці, що раніше було характерно тільки для клієнтської сторони. Це робить Node.js ідеальним вибором для побудови масштабованих серверних додатків, які можуть обробляти одночасні запити і забезпечувати високу продуктивність.

Express.js — мінімалістичний і гнучкий фреймворк для Node.js, який забезпечує широкий набір інструментів для розробки веб-додатків і API.

Однією з головних причин використання Express.js є його простота і швидкість розробки.

Хорошою перевагою Express.js є його модульна структура, яка дозволяє додавати необхідні функції через middleware — функції, які обробляють запити і відповіді. Їх можна використовувати для виконання завдань, таких як обробка файлів cookie, парсинг JSON, логування запитів, аутентифікація і авторизація. Це робить Express.js дуже гнучким і дозволяє легко налаштовувати додаток відповідно до потреб проєкту.

2.1.3. React.js

React — це бібліотека JavaScript, розроблена Facebook, яка використовується для побудови користувацьких інтерфейсів, особливо для SPA.

Головною особливістю React є його компонентний підхід, який дозволяє розробникам створювати складні інтерфейси, розбиваючи їх на багато незалежних, повторно використовуваних частин. Кожен компонент у React інкапсулює власну логіку та рендеринг, що спрощує розробку і підтримку великомасштабних додатків.

Однією з ключових переваг React є віртуальний DOM для оптимізації процесу оновлення інтерфейсу користувача. Коли стан додатка змінюється, React оновлює віртуальний DOM, порівнює його з попереднім станом і лише тоді оновлює реальний DOM, роблячи мінімальні зміни. Це значно підвищує продуктивність застосунку, зменшуючи кількість операцій з реальним DOM, які є відносно повільними.

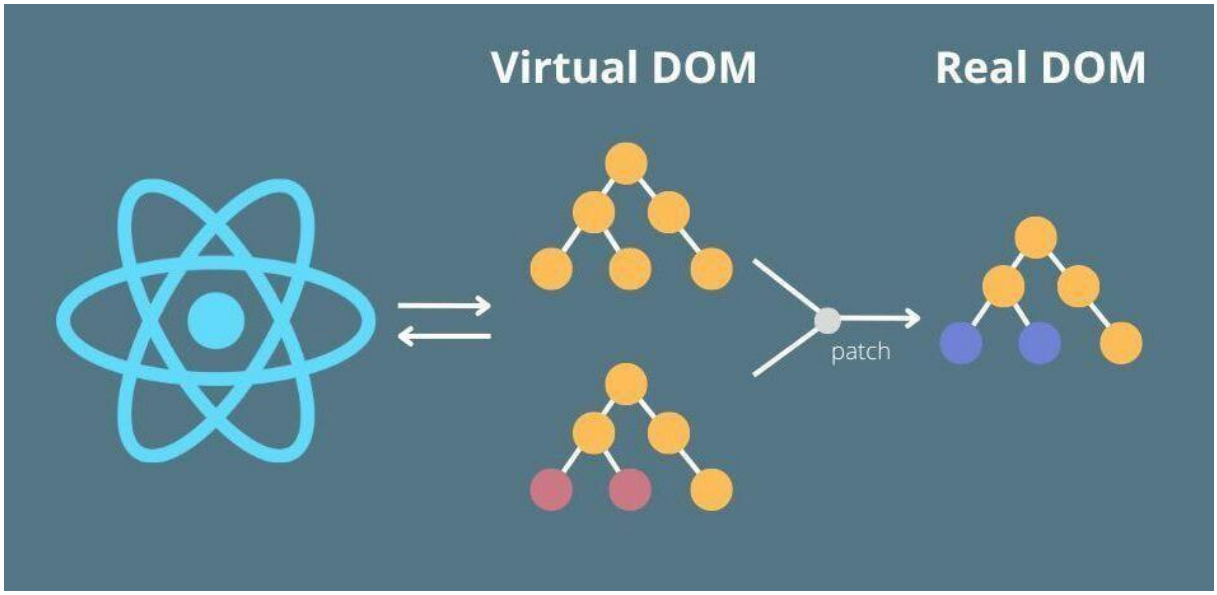


Рис. 2.2. Візуалізація змін у DOM елементі з використанням віртуальної DOM

Екосистема React включає потужні інструменти, такі як Router для керування маршрутизацією і Redux для управління станом додатка. Ці інструменти розширюють можливості React і роблять його ще більш ефективним.

React також забезпечує односторонній потік даних, які передаються від батьківських компонентів до дочірніх. Це допомагає краще контролювати стан додатка і робить його передбачуваним, полегшуючи відладку і тестування. React використовує JSX, синтаксичне розширення JavaScript, яке дозволяє розробникам писати компоненти, використовуючи HTML-подібний синтаксис, що робить код більш зрозумілим і читабельним.

2.1.4. Redux.js

Redux.js — це бібліотека для керування станом в JavaScript-застосунках, особливо корисна для додатків з великою кількістю взаємодій між компонентами. Вона була створена для вирішення проблеми з передаванням стану та управлінням ним у складних додатках, так званої props drilling.

Для продуктивної роботи з цією бібліотекою та уникнення її непередбачуваної поведінки, варто дотримуватись трьох основних принципів:

- Весь стан додатка має зберігатися в одному об'єкті дерева станів (store).
- Зміни в стан повинні вноситись тільки через дію (action) — об’єкт з обов’язковим полем type, який описує що змінилося.
- Щоб визначити, як стан змінюється у відповідь на дії, потрібно писати чисті функції, які називаються ред'юсерами (reducers). Ред'юсери приймають попередній стан і дію, а повертають новий стан.

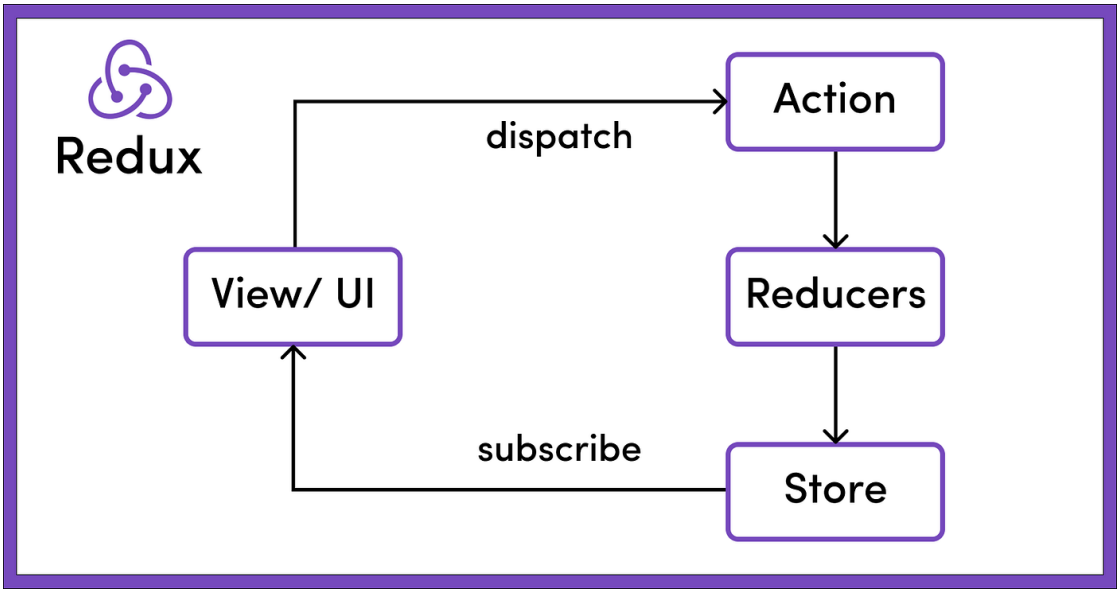


Рис. 2.3. Принцип роботи Redux

2.1.5. SCSS

SCSS (Sassy CSS) — це розширення для CSS, що додає нові можливості та полегшує роботу з каскадними таблицями стилів. В основі SCSS лежить синтаксично вірна мова SASS, яка дозволяє використовувати змінні, вкладені правила, міксини, наслідування і багато інших функцій, що роблять код стилів більш організованим і легким для підтримки.

Використовуючи SCSS, можна створювати більш динамічні та потужні стилі. Це сприяє зменшенню кількості дубльованого коду і підвищенні його гнучкості.

Однією з головних переваг SCSS є можливість використання змінних, що дозволяє зберігати повторювані значення, такі як кольори, шрифти або розміри, в одному місці і легко змінювати їх по всьому проєкту. Це особливо корисно при створенні великих проєктів, де такі значення можуть використовуватися у багатьох різних місцях.

Міксини та функції в SCSS також дозволяють створювати багаторазові блоки стилів, які можна викликати за потреби, роблячи код чистішим і менш схильним до помилок.

```scss
@mixin pageGrid {
  --columns: 4;
  display: grid;
  column-gap: 16px;
  grid-template-columns: repeat(var(--columns), 1fr);

  @include onTablet {
    --columns: 12;
  }

  @include onDesktop {
    --columns: 24;
  }
}
```

Рис. 2.4. Приклад задання міксина

```scss
.cart {
  @include pageGrid;
  row-gap: 32px;
  font-size: 32px;
  line-height: 41px;

  &__title{
    grid-column: span 4;
    margin: 0;
    font-size: 32px;
    line-height: 41px;

    @include onTablet {
      grid-column: span 12;
      font-size: 48px;
      line-height: 56px;
    }

    @include onDesktop {
      grid-column: span 24;
    }
  }
```

Рис. 2.5. Приклад використання міксинів та вкладеності SCSS

Вкладеність у SCSS забезпечує більш природний і зрозумілий спосіб написання стилів, відображаючи ієрархію HTML-коду. Це дозволяє легко бачити, які стилі застосовуються до яких елементів і як вони взаємодіють між собою. Такий підхід значно підвищує читабельність і зручність роботи з кодом, особливо в проєктах зі складною структурою.

Наслідування в SCSS дає змогу одному селектору успадковувати стилі іншого, що спрощує написання і підтримку CSS-коду. Це дозволяє уникнути дублювання стилів і забезпечує більш чистий та організований код.

2.2. Проєктування архітектури

Проєктування архітектури є важливим етапом розробки будь-якого веб-додатку, оскільки від нього залежить ефективність, масштабованість і підтримуваність системи. Вибір відповідної архітектури дозволяє розподілити обов'язки між компонентами додатку, що забезпечує чітку структуру та легкість в управлінні кодовою базою.

MVC архітектура є одним з найпоширеніших шаблонів проєктування в розробці програмного забезпечення. Вона розділяє додаток на три основні компоненти: Модель (Model), Представлення (View) та Контролер (Controller). Цей поділ допомагає ізолювати бізнес-логіку від інтерфейсу користувача, забезпечуючи чітке розділення обов'язків та сприяючи легшій підтримці та масштабованості коду.

Модель відповідає за управління даними та бізнес-логікою. Вона обробляє запити на збереження, оновлення, видалення та отримання даних.

Модель безпосередньо взаємодіє з базою даних або іншими джерелами, що гарантує доступ до інформації, необхідної для роботи додатку. Важливою особливістю моделі є те, що вона не має уявлення про інтерфейс користувача або контролерів, зберігаючи свою функціональність незалежною від інших компонентів.

Представлення відповідає за відображення даних, які отримує від моделі, користувачу. Це може бути веб-сторінка, мобільний інтерфейс або будь-який інший тип інтерфейсу користувача. View отримує інформацію від контролера і формує остаточний вигляд, який буде показаний користувачу.

Представлення також може включати логіку відображення, таку як форматування дат або чисел, але не повинно містити бізнес-логіку. Цей

компонент тісно пов'язаний з контролером, який надає йому необхідні дані для відображення.

Контролер діє як посередник між моделлю і представленням. Він обробляє вхідні запити від користувача, взаємодіє з моделлю для виконання необхідних операцій та обирає відповідне представлення для відображення результатів. Контролер приймає рішення про те, як обробляти запити, і виконує відповідні дії, такі як виклик методів моделі для отримання або зміни даних. Контролер також може виконувати перевірку даних, обробку помилок та інші завдання, пов'язані з логікою додатку.

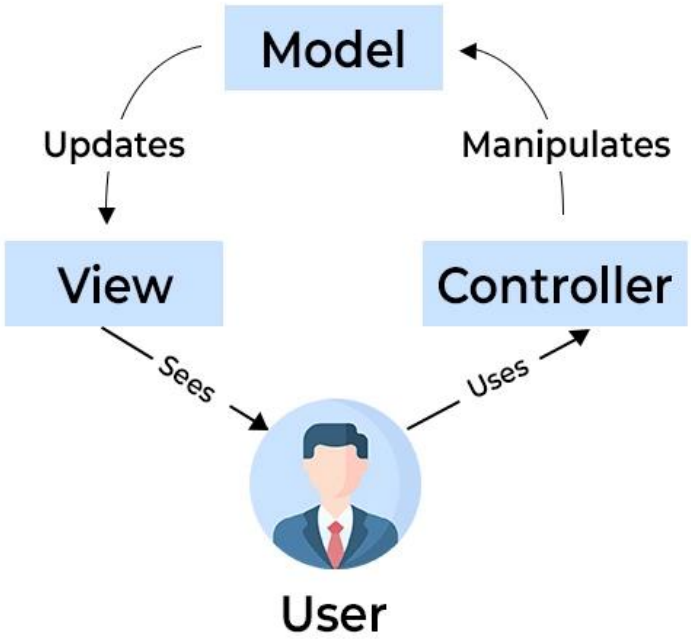


Рис. 2.6. Принцип роботи компонентів MVC

Незважаючи на деякі складнощі, пов'язані з початковою настройкою та можливим перевантаженням контролерів, переваги MVC значно переважають її недоліки, роблячи цей підхід одним з найбільш популярних та широко використовуваних у сучасній розробці програмного забезпечення.

Feature-Sliced Design (FSD) є сучасним підходом до організації коду в фронтенд-розробці, орієнтованим на розділення функціональностей на ізольовані, незалежні частини. Цей підхід сприяє кращій модульності, підтримуваності та масштабованості проєкту, дозволяючи розробникам ефективніше управляти складністю великомасштабних веб-додатків.

Кожна частина додатку (фіча) розробляється як самостійний модуль, що включає всі необхідні компоненти для її реалізації: логіку, дані, стилі та тести.

Slices (сегменти) представляють собою великі функціональні блоки додатку, кожен з яких відповідає за окрему частину бізнес-логіки. Наприклад, в інтернет-магазині такими сегментами можуть бути "Каталог товарів", "Кошик", "Замовлення". Кожен сегмент містить компоненти, які виконують специфічні для нього функції.

Features (функції) - це менші частини, що складають сегменти, і відповідають за конкретні завдання або функціональні можливості. Наприклад, функція в сегменті "Каталог товарів" може бути відповідальною за відображення списку товарів, пошук або фільтрацію. Кожна функція містить власні компоненти, які реалізують її логіку, інтерфейс і стиль.

Entities (сутності) представляють базові об'єкти доменної логіки, з якими працюють функції. Це можуть бути моделі даних, такі як продукти, користувачі або замовлення. Сутності організовані так, щоб їх можна було легко повторно використовувати в різних частинах додатку.

Shared (загальні компоненти) включають загальні елементи, які використовуються у різних частинах додатку. Це можуть бути базові компоненти інтерфейсу, утиліти, хелпери або стилі, які не належать до конкретної функції чи сегменту, але використовуються у багатьох з них.

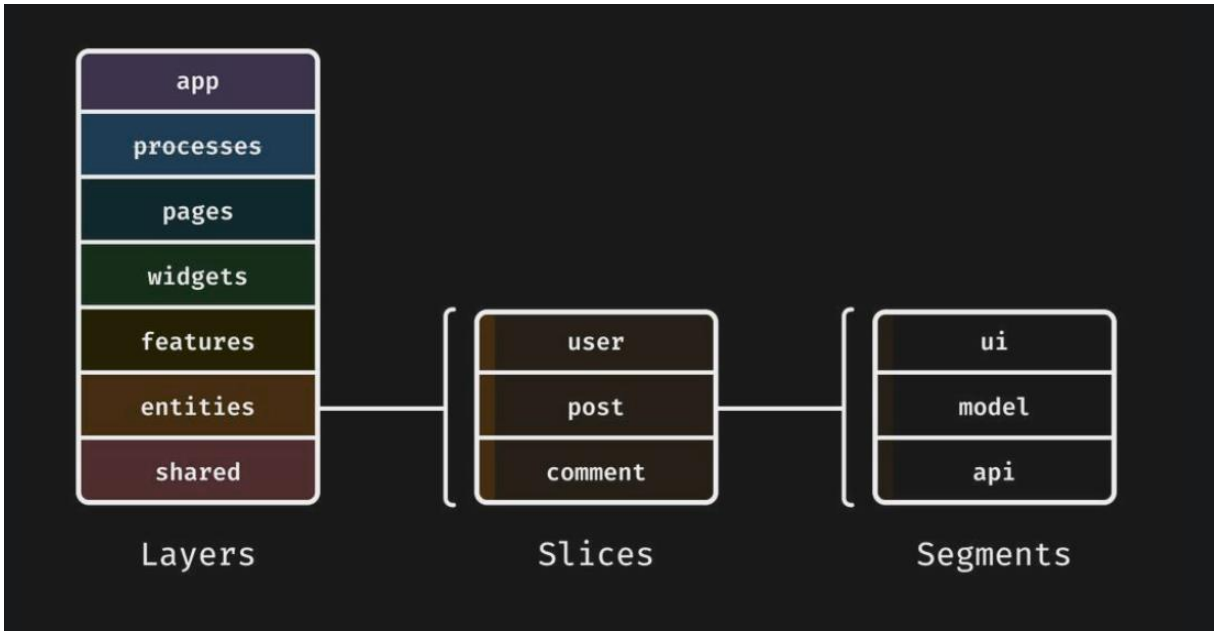


Рис. 2.7. Схема архітектури FSD

2.3. Структура бази даних

PostgreSQL — це потужна, відкрита об'єктно-реляційна система управління базами даних (СУБД), яка підтримує більшість стандартів SQL і забезпечує широкий набір функцій для роботи з базами даних.

Основні можливості:

- Відкритий код: PostgreSQL є безкоштовною і з відкритим кодом, що дозволяє користувачам змінювати, розширювати і використовувати її без обмежень.
- Підтримка стандартів SQL: підтримка більшості стандартів SQL, включаючи складні запити, транзакції, зовнішні ключі, тригери, перегляди та збережені процедури.
- Розширюваність: можливість додавати нові типи даних, функції, операції і методи індексації, що робить її дуже гнучкою.

- Підтримка ACID: PostgreSQL забезпечує високий рівень надійності і консистентності даних завдяки повній підтримці ACID-транзакцій.
- Реплікація і відновлення: наявність реплікації даних, що дозволяє створювати резервні копії і забезпечувати високу доступність бази даних.

PostgreSQL використовує клієнт-серверну архітектуру. Сервер, відомий як постмайстер, відповідає за обробку запитів до бази даних, управління транзакціями і забезпечення цілісності даних. Клієнти підключаються до сервера через мережу і відправляють SQL-запити для виконання.

База даних інтернет-магазину містить наступні таблиці:

Categories

Зберігає інформацію про різні категорії товарів, що пропонуються в інтернет-магазині. Стовпці: id (унікальний ідентифікатор категорії) і name (назва категорії).

Colors

Містить дані про доступні кольори товарів. Стовпці: id (унікальний ідентифікатор кольору), name (назва кольору) і hex (код кольору).

Descriptions

Містить детальні описи товарів. Стовпці: id (унікальний ідентифікатор опису), product_id (ідентифікатор товару), title (заголовок опису) і description (текст опису).

Discounts

Зберігає інформацію про доступні знижки. Стовпці: id (унікальний ідентифікатор знижки) і value (значення знижки).

Images

Містить дані про зображення товарів. Стовпці: id (унікальний ідентифікатор зображення), product_id (ідентифікатор товару) і string (посилання на зображення).

Orders

Зберігає інформацію про замовлення. Стовпці: id (унікальний ідентифікатор замовлення) і owner_id (ідентифікатор користувача).

Product_Color

Використовується для зв'язку товарів з їх доступними кольорами. Стовпці: product_id і color_id.

Products

Містить основну інформацію про товари.

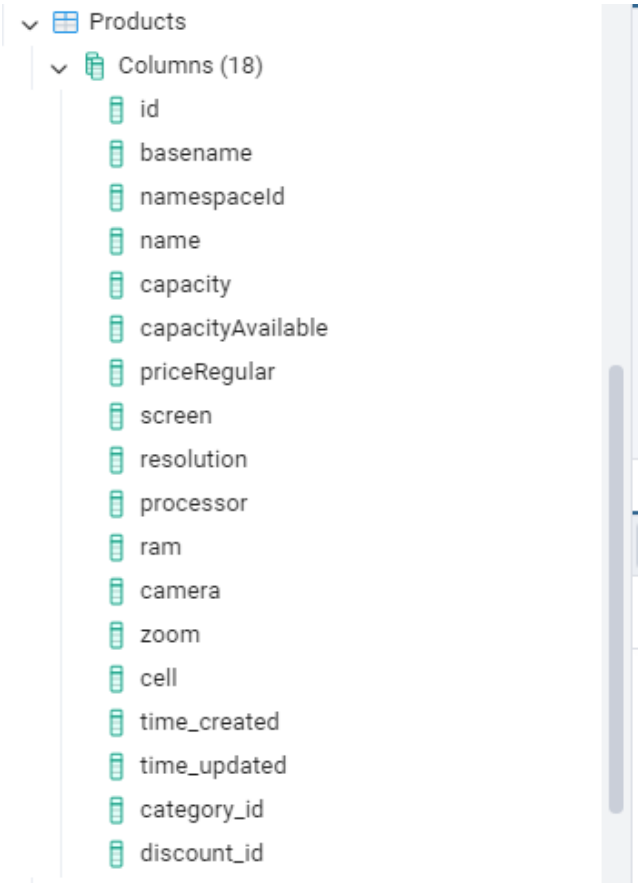


Рис. 2.8. Наповнення таблиці Products

2.4. Реалізація Backend частини веб-застосунку

Першим кроком у розробці будь-якого веб-додатку є налаштування робочого середовища, що включає у себе встановлення потрібних фреймворків та бібліотек. Важливо встановити відповідні залежності проєкту для коректної роботи інструментів.

```
"dependencies": {
    "@types/cookie-parser": "^1.4.6",
    "convert-css-color-name-to-hex": "^0.1.1",
    "cookie-parser": "^1.4.6",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "express-paginate": "^1.0.2",
    "express-validator": "^7.0.1",
    "js-md5": "^0.8.3",
    "pg": "^8.11.3",
    "pg-hstore": "^2.3.4",
    "reflect-metadata": "^0.1.14",
    "sequelize": "^6.35.1",
    "sequelize-cli": "^6.6.2",
    "sequelize-typescript": "^2.1.6",
    "slugify": "^1.6.6",
    "ts-node": "^10.9.2",
    "uuid": "^9.0.1"
  },
  "devDependencies": {
    "@types/cors": "^2.8.17",
    "@types/express": "^4.17.21",
    "@types/node": "^20.10.4",
    "@types/validator": "^13.11.7",
    "nodemon": "^3.0.2",
    "typescript": "^5.3.2"
  }
```

Рис. 2.9. Залежності у файлі package.json

Для зручної роботи з базою даних через об'єктно-орієнтований інтерфейс було вирішено використати Sequelize — ORM (Object-Relational Mapping) бібліотеку. Вона надає можливості визначення моделей і їх відносин, валідацію даних, автоматичне створення SQL запитів із зручним API, підтримку транзакцій та безпеку.

src > models > TS Category.model.ts > ...

```typescript
import {Table, Column, Model, HasOne} from 'sequelize-typescript';
import {DataTypes} from "sequelize";

import Product from "./Product.model";

@Table({
    modelName: 'Category',
    tableName: 'Categories',
    timestamps: false
})
export default class Category extends Model {
    @Column({
        type: DataTypes.UUID,
        defaultValue: DataTypes.UUIDV4,
        primaryKey: true,
        allowNull: false
    })
    id: string;

    @Column({
        type: DataTypes.STRING,
        allowNull: false
    })
    name: string;

    @HasOne(() => Product, 'category_id')
    product: Product
}

```

Рис. 2.10. Модель таблиці Categories

```
export const get = async (req: Request, res: Response) => {
    try {
        const {
            category = req.baseUrl.slice(1),
            query = '',
```

```
            page = 0,
            limit = 10,
            sortBy,
            desc = false
        } = req.query
        const offset = Math.ceil(+page * +limit)
        const sort: { field?: string, isDESC?: boolean } = {}

        if (!category) {
            return res.sendStatus(400)
        }

        if (sortBy) {
            sort.field = SortFields[sortBy.toString()]
            sort.isDESC = (desc === 'true')
        }

        const products = await ProductService.get({
            category: category.toString(),
            limit: +limit,
            offset,
            sort,
            filters: {
                query: query.toString()
            }
        })

        if (products) {
            const paginationNav = {
                nextPage: Math.floor(offset/+limit) + 1,
                prevPage: Math.floor(offset/+limit) - 1 >= 0 ?
Math.floor(offset/+limit) - 1 : null
            }
            const resBody: { [key: string]: any } = {
                total: products.count,
                onPage: products.rows.length,
                nextPage: null,
                prevPage: null,
                data: products.rows,
            }

            if (products.rows.length >= +limit) {
```

```
                const nextSearchParams = new URLSearchParams()

                nextSearchParams.append('limit', limit.toString())
                nextSearchParams.append('page',
paginationNav.nextPage.toString())
                if (query) {
                    nextSearchParams.append('query', query.toString())
                }

                resBody.nextPage =
`/${category}?${nextSearchParams.toString()}`
            }

            if (paginationNav.prevPage !== null) {
                const prevSearchParams = new URLSearchParams()

                prevSearchParams.append('limit', limit.toString())
                if (paginationNav.prevPage > 0) {
                    prevSearchParams.append('page',
paginationNav.prevPage.toString())
                }
                if (query) {
                    prevSearchParams.append('query', query.toString())
                }

                resBody.prevPage  =
`/${category}?${prevSearchParams.toString()}`
            }

            return res.send(resBody)
        }
    }
    catch (e) {
        return res.sendStatus(500)
    }
}
```

Дана функція належить контролеру продукта та реалізує обробку запиту типу GET для отримання продуктів з бази даних з урахуванням різних параметрів, переданих через query URL. Він витягує параметри, такі як категорія,

запит, сторінка та ліміт, обчислює зміщення для пагінації, встановлює сортування за вказаними параметрами, і передає їх до ProductService для отримання відповідних продуктів. Якщо продукти успішно знайдені, формує відповідь, яка включає загальну кількість продуктів, дані сторінки та посилання на наступну і попередню сторінки з урахуванням пагінації. У разі помилки взаємодії з базою даних або сервісом продуктів, відправляє відповідь з HTTP статусом 500.

```
export const get = async ({ filters, limit = 10, offset = 0, category,
sort }: QueryParams) => {
    try {
        let whereStatement: WhereStatement = {}
        let OrderStatement: OrderItem[] = []

        if (filters?.query) {
            whereStatement.name = {
                [Op.iLike]: `%${filters.query}%`
            }
        }

        if (filters?.byDiscount) {
            OrderStatement.push([{ model: Discount, as: 'discount' },
'value', "DESC"])
        }

        if (filters?.byDate) {
            OrderStatement.push(['time_created', "DESC"])
        }

        if (sort?.field) {
            OrderStatement.push([sort.field, sort.isDESC ? "DESC" :
"ASC"])
        }

        return Product.findAndCountAll({
            include: [
                {
                    model: Category,
                    attributes: ['name'],
```

```
                    where: {
                        name: category,
                    }
                },
                {
                    model: Image,
                    attributes: ['string'],
                    limit: 1,
                },
                {
                    model: Discount,
                    as: 'discount',
                    attributes: ['value'],
                }
            ],
            where: whereStatement,
            attributes: {
                exclude: ['category_id', 'discount_id']
            },
            limit,
            offset,
            order: OrderStatement
        })
    } catch (e) {
        return Promise.reject("Server error")
    }
}
```

Це функція get у сервісі, яка призначена для отримання продуктів з бази даних з урахуванням параметрів, що надійшли з контролера. Вона формує SQL запит, який включає в себе умови пошуку (за наявності), зовнішні зв'язки з моделями Category, Image та Discount, і відповідні атрибути для відображення результатів.

```
export const getByNamespaceId = async (req: Request, res: Response) => {
    try {
```

```
        const { namespaceId } = req.params
        const product = await ProductService.getByNamespaceId({
namespaceId })

        if (product) {
            return res.send(product)
        }

        return res.sendStatus(404)
    } catch (e) {
        return res.sendStatus(500)
    }
}
```

Ця функція контролера getByNamespaceId призначена для обробки HTTP GET-запитів, спрямованих на отримання даних конкретного продукту за його унікальним ідентифікатором. Вона спочатку отримує namespaceId з параметрів запиту, потім викликає відповідну функцію сервісу getByNamespaceId для отримання продукту з бази даних. Якщо продукт знайдено, він відправляється у відповідь за допомогою res.send(product). Якщо продукт не знайдено, повертається статус 404 "Not Found". У разі виникнення помилки під час обробки запиту (наприклад, помилка бази даних), відправляється статус 500 "Internal Server Error".

```
export const getByNamespaceId = async ({ namespaceId }: { namespaceId:
string }) => {
    try {
        const product = await Product.findOne({
            include: [
                {
                    model: Image,
                    attributes: ['id', 'string']
                },
                {
                    model: Description,
                    attributes: { exclude: ['product_id'] }
```

```
                },
                {
                    model: Discount,
                    attributes: ['value'],
                },
                {
                    model: Color,
                    through: {
                        attributes: [],
                    }
                }
            ],
            where: { namespaceId },
            attributes: {
                exclude: ['category_id', 'discount_id']
            },
        })

        return product
    } catch (e) {
        return Promise.reject("Server error")
    }
}
```

Функція сервісу getByNamespaceId, використана у відповідній функції контролера продукту.

```
export const getRecommended = async (req: Request, res: Response) => {
    try {
        const { namespaceId } = req.params
        const { limit = 10 } = req.query
        const product = await ProductService.getByNamespaceId({
namespaceId })

        if (product) {
            const recommended = await ProductService.getRecommended({
                basename: product.basename,
                limit: +limit
            })
```

```
            return res.send(recommended)
        }

        return res.sendStatus(404)
    } catch (e) {
        return res.sendStatus(500)
    }
}
```

Ця функція контролера getRecommended відповідає за обробку запитів для отримання рекомендованих продуктів на основі ідентифікатора, отриманого з параметрів запиту. Спочатку вона викликає функцію сервісу getByNamespaceId для отримання даних про конкретний продукт. Якщо продукт знайдено, вона викликає getRecommended сервісу, яка виконує запит до бази даних за допомогою моделі Product, обираючи всі продукти, що мають вказаний basename. Якщо запит виконується успішно, функція повертає знайдені продукти.

```
export const getRecommended = async ({ basename, limit }: { basename:
string, limit: number }) => {
    try {
        return Product.findAll({
            where: { basename },
            include: [
                {
                    model: Category,
                    attributes: ['name'],
                },
                {
                    model: Image,
                    attributes: ['string'],
                    limit: 1,
                },
                {
                    model: Discount,
                    as: 'discount',
                    attributes: ['value'],
```

```
                }
            ],
            attributes: {
                exclude: ['category_id', 'discount_id']
            },
            limit,
        })
    } catch (e) {
        return Promise.reject("Server error")
    }
}
```

Функція сервісу getRecommended, яка викликається у відповідному контролері.

```
import express from 'express';
import {get, getByNamespaceId, getRecommended, getSpecials} from
'../controllers/Product.controller'

const router = express.Router();

router.use(express.json());

router.get('/', get);

router.get('/:namespaceId', getByNamespaceId);

router.get('/:namespaceId/recommended/', getRecommended);

router.get('/specials/latest', getSpecials('latest'));

router.get('/specials/hot-price', getSpecials('discount'));

export default router
```

Роутер встановлює відповідність між URL-адресами і функціями обробки, які викликаються при запитах до цих адрес. Кожен маршрут налаштований для обробки відповідних типів запитів і спрямовує їх до функцій контролерів.

2.5. Реалізація Frontend частини веб-застосунку

Архітектурою клієнтської частини додатка є FSD, тому кожна папка має специфічне призначення.

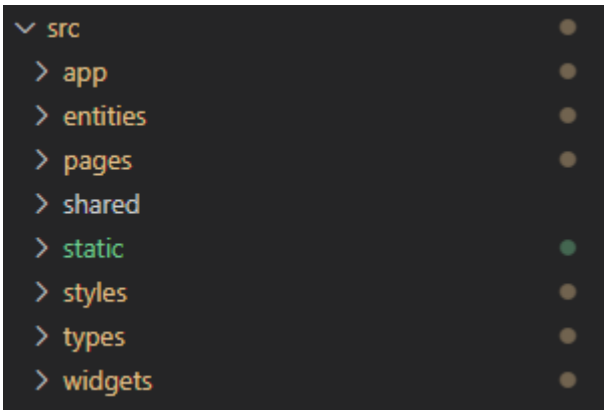


Рис. 2.11. Структура додатка

- src: коренева папка проєкту, що містить весь вихідний код додатку.
- app: містить загальні налаштування і конфігурації додатку, такі як створення стору Redux, глобальні провайдери і конфігурації маршрутизації.
- entities: вміщує логіку та компоненти, що представляють основні сутності додатку.
- pages: містить компоненти сторінок додатку. Кожна сторінка може об'єднувати різні віджети, блоки та інші компоненти для відображення контенту.
- shared: вміщує загальні компоненти, утиліти та допоміжні функції, які можуть використовуватися в різних частинах додатку. Наприклад кнопки, модальні вікна, хелпери та інше.
- static: зберігає статичні файли, такі як зображення, іконки та інші ресурси, що не потребують компіляції.

- styles: вміщує глобальні стилі додатку, файли зі змінними та міксинами для стилізації компонентів.
- types: містить типи TypeScript, що використовуються у проєкті для типізації даних та функцій.
- widgets: вміщує компоненти, що складаються з інших елементів або забезпечують специфічний функціонал, який може використовуватись на різних сторінках додатку.

Рис. 2.12. Залежності Frontend частини веб-додатку

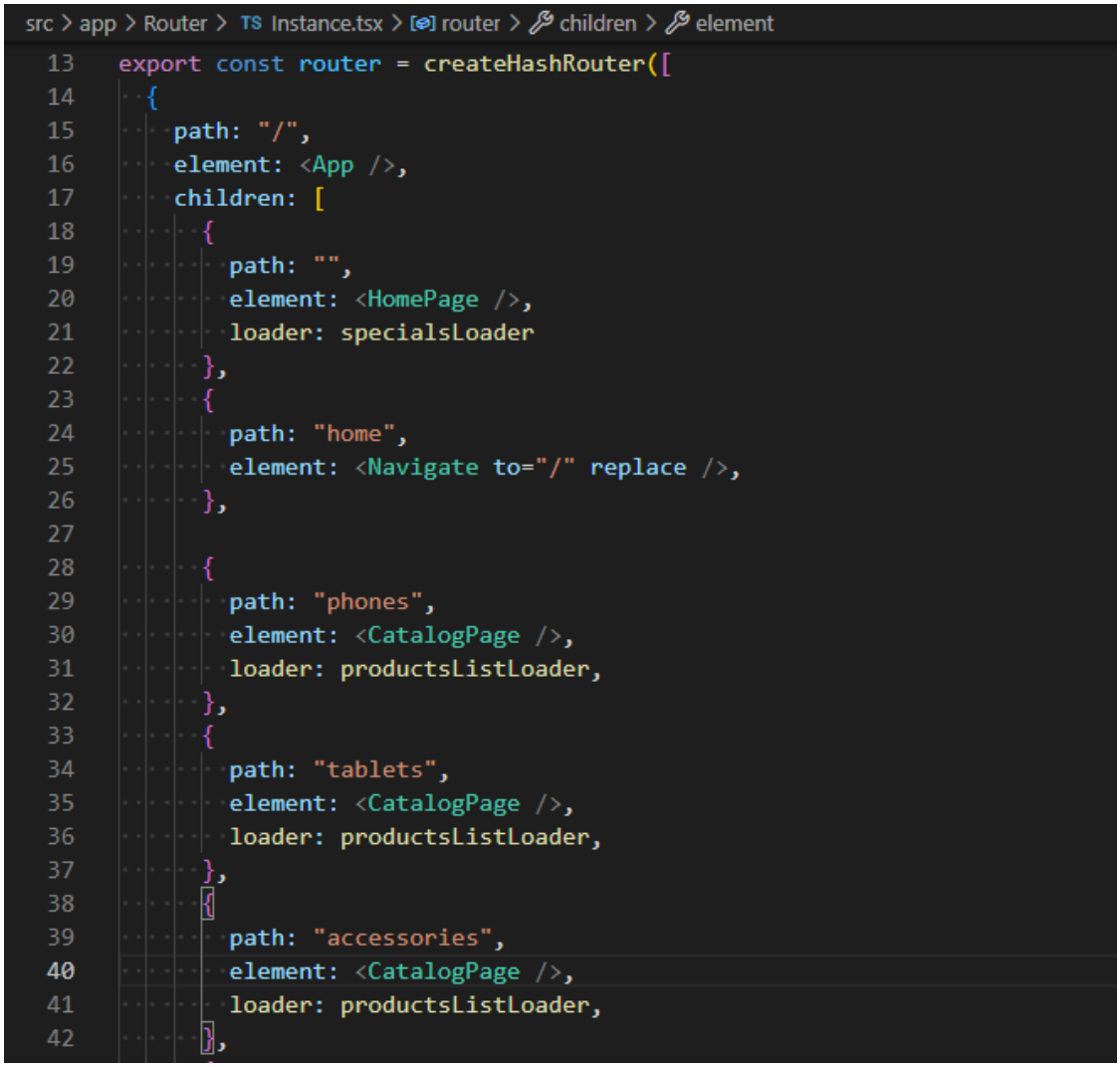


Рис. 2.13. Маршрутизація

React Router — це бібліотека для керування маршрутизацією. Вона підтримує динамічну маршрутизацію, вкладені роути, маршрути з параметрами та забезпечує компоненти для роботи з історією браузера.

src > app > TS App.tsx > ...

```tsx
import { Outlet, useLocation } from "react-router-dom";
import Header from "../widgets/Header/Header";
import Footer from "../widgets/Footer/Footer";
import styles from "../styles/main.module.scss";
import Bredcrumbs from "../shared/ui/Bredcrumbs/Bredcrumbs";
import { ThemeProvider } from "./providers/ThemeProvider";

export const App = () => {
  const location = useLocation();
  const pathname = location.pathname;

  return (
    <ThemeProvider>
      <div className={styles.appWrapper}>
        <Header />
        <main className={styles.mainWrapper}>
          {pathname !== "/" && pathname !== "/cart" && <Bredcrumbs />}
          <div className={styles.mainContent}>
            <Outlet />
          </div>
        </main>
        <Footer />
      </div>
    </ThemeProvider>
  );
};

```

Рис. 2.14. Кореневий компонент додатку

Тут знаходиться увесь застосунок. Обгортка, Header та Footer присутні на всіх сторінках додатку. Компонент Outlet відповідає за наповнення сторінки, він змінюється динамічно в залежності від обраного маршруту. Наприклад на сторінці CategoryProducts – Outlet буде вміщати в собі наповнення каталогу.

src > pages > HomePage > TS HomePage.tsx > HomePage

```tsx
export const HomePage = () => {
  const { latest, hotPrice } = useLoaderData() as {
    latest: Promise<ProductShorted>;
    hotPrice: Promise<ProductShorted>;
  };

  return (
    <>
      <h1 className={styles.heading}>Welcome to Apple Box store!</h1>

      <Carousel />

      <AsyncWrapper
        data={latest}
        Loader={<ItemSlider name="Brand new model" state="loading" />}
        Error={<ItemSlider name="Brand new model" state="error" />}
      >
        <ItemSlider name="Brand new model" />
      </AsyncWrapper>

      <Category />

      <AsyncWrapper
        data={hotPrice}
        Loader={<ItemSlider name="Hot prices" state="loading" />}
        Error={<ItemSlider name="Hot prices" state="error" />}
      >
        <ItemSlider name="Hot prices" />
      </AsyncWrapper>
    </>
  );
```

Рис. 2.15. Головна сторінка

useLoaderData використовується для завантаження даних до компоненту HomePage. Це частина бібліотеки react-router-dom, яка дозволяє завантажувати дані для маршруту перед тим, як компонент буде відрендерений.

Компонент AsyncWrapper слугує обгорткою для інших компонентів. Він відповідає за стан відображення змісту:

- data: контент, що відображається після отримання даних із сервера.
- Loader: тимчасовий стан, активний тоді, коли дані з API ще не прийшли.
- Error: дані не дійшли із-за певної помилки.

```
const settings = {
    dots: true,
    infinite: true,
    autoplay: true,
    arrows: true,
    speed: 500,
    slidesToShow: 1,
    autoplaySpeed: 5000,
    slidesToScroll: 1,
    responsive: [
      {
        breakpoint: 640,
        settings: {
          arrows: false,
        }
      }
    ]
  };
  return (
    <div>
      <Slider
        {...settings}
        className={`${styles.carousel} ${
          theme === "light"
            ? "carousel_main carousel_main_light"
            : "carousel_main"
        }`}
      >
        {banners.map((banner, index) => (
          <Link
            to={banner.link}
            key={index}
            className={styles.carousel_wrap_item}
            onClick={() => scrollToTop()}
          >
            <img
              className={styles.carousel_banner}
              src={isSmallScreen ? banner.mobile : banner.desktop}
              alt={`banner${index + 1}`}
              />
            </Link>
        ))}
      </Slider>
    </div>
  );
};
```

Рис. 2.16. Компонент “карусель”

```
    <div
      className={`${styles.sliderContainer} ${
        theme === "light"
          ? "card_slider card_slider_light"
          : "card_slider"
      }`}
    >
      <h2 className={styles.sliderTitle}>{name}</h2>
      <Slider {...settings}>
        {state
          ? [1, 2, 3, 4, 5].map(id => <ItemCardState state={state} key={id} />)
          : data.map(phone => <ItemCard phone={phone} key={phone.id} />)}
      </Slider>
    </div>
  );
};
```

Рис. 2.17. Компонент слайдер

Для реалізації цих віджетів було використано зовнішню бібліотеку React Slick. Вона надає можливості створення динамічних та інтерактивних карусельних слайдерів зі значними можливостями налаштування. React Slick дозволяє легко інтегрувати каруселі в додатки React, надаючи API для керування подіями, зображеннями, слайдами та іншими параметрами каруселі.

Компонент CategoryProducts див. [Додаток А] є найбільш громіздким, адже у ньому присутні товари, сортування, пошук та пагінація. Залежно від параметру в URL адресі, а саме категорії, на сторінці відображаються відповідні гаджети.

```
42    &__title{
43      grid-column: span 4;
44      margin: 0;
45      font-size: 32px;
46      line-height: 41px;
47
48      @include onTablet {
49        grid-column: span 12;
50        font-size: 48px;
51        line-height: 56px;
52      }
53
54      @include onDesktop {
55        grid-column: span 24;
56      }
57    }
58
```

Рис. 2.18. Використання міксинів для адаптивної верстки

```
export const get: GetProducts = async ({
  category,
  page,
  limit,
  query,
  sortBy,
  desc
}) => {
  const searchParams = new URLSearchParams();

  if (page) {
    searchParams.set("page", page);
  }

  if (limit) {
    searchParams.set("limit", limit);
  }

  if (query) {
    searchParams.set("query", query);
  }

  if (sortBy) {
    searchParams.set("sortBy", sortBy);
  }

  if (desc) {
    searchParams.set("desc", desc);
  }

  const response = await instance.get(`/${category}?${searchParams.toString()}`);

  if (response.status === 200) {
    const { total, onPage, nextPage, prevPage, data } = response.data;

    return { total, onPage, nextPage, prevPage, data: data.map((item: Product) => {
      delete item.discount;

      if (item.images) {
        const image = item.images[0].string;
        delete item.images;

        return { ...item, image };
      }
      return { ...item };
    }) };
```

Рис. 2.19. Приклад “спілкування” із сервером

Цей компонент get виконує HTTP запит до сервера для отримання продуктів із вказаними параметрами, такими як категорія, сторінка, ліміт, запит для пошуку, сортування та напрямок сортування. Формується запит з використанням параметрів URL, які передаються у вигляді параметрів до API сервера.

```css
:root,
[data-theme="dark"] {
  --accent-color: #8D6262;
  --accent-color-hov: #ED8D8D;

  --primary-text-color: #f1f2f9;
  --secondary-text-color: #7f757b;

  --separator-color: #8D6262;

  --bg-color-main: #393232;
  --bg-color-product: #4D4545;
  --bg-color-controllers: #8D6262;
  --bg-color-controllers-hov: #ED8D8D;

  --btn-color-text: #f1f2f9;
  --btn-inactive-color-text: #f1f2f9;
  --error_color: #eb5757;
  --success_color: #27ae60;
}

[data-theme="light"] {
  --accent-color: #f86800;
  --accent-color-hov: #f98837;

  --primary-text-color: #0f0f11;
  --secondary-text-color: #89939a;

  --separator-color: #e2e6e9;

  --bg-color-main: #F6F5F2;
  --bg-color-product: #f8f8f8;
  --bg-color-controllers: #e9ecee;
  --bg-color-controllers-hov: #f1f5f8;

  --btn-color-text: #f1f2f9;
  --btn-inactive-color-text: #0f0f11;
  --error_color: #f86800;
  --success_color: #27ae60;
}
```

Рис. 2.20. Реалізація зміни колірної теми додатка

Увесь додаток огортає ThemeProvider. Змінні CSS мають різний колір в залежності від того, якій глобальній змінній, themeDark чи themeLight, прирівнюється змінна theme із контексту. Іконки можна підбирати за кольором використовуючи тернарний оператор.

РОЗДІЛ 3. ІНТЕРФЕЙС КОРИСТУВАЧА. ІНСТРУКЦІЯ ДЛЯ ЗАСТОСУВАННЯ

У будь-якому веб-застосунку найбільш важливим елементом інтерфейсу є безперечно навігація. Header поділений на дві частини. Зліва розміщені посилання на головну сторінку і каталоги товарів. Справа – кнопка зміни колірної теми додатку, посилання на вподобані товари і кошик.

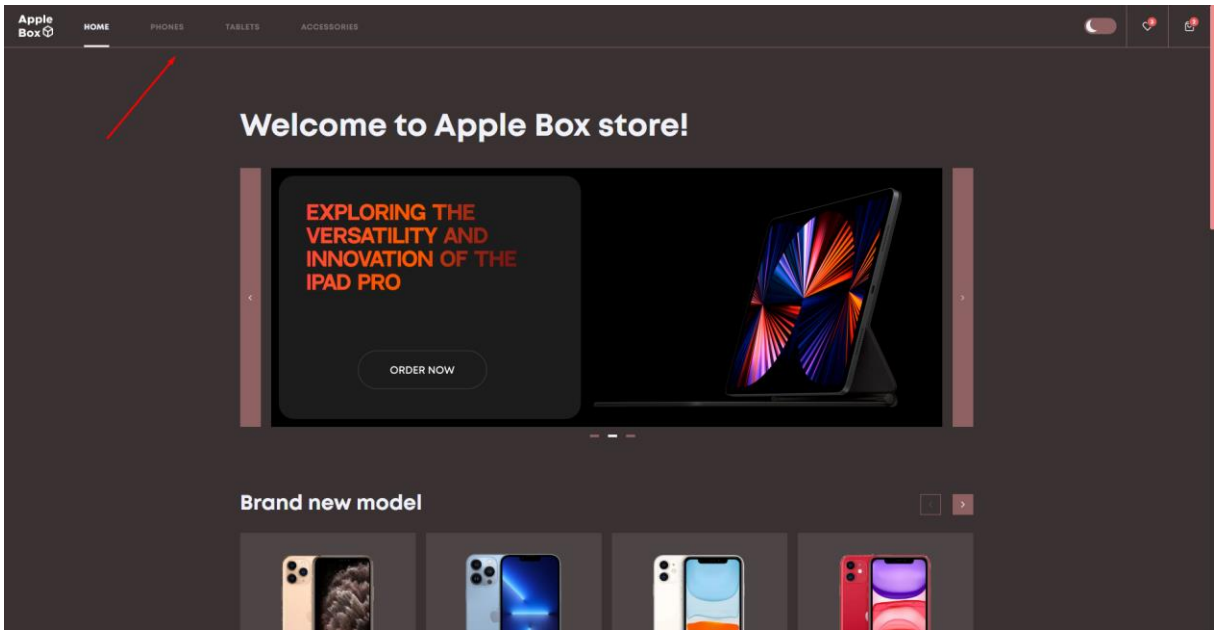


Рис. 3.1. Інтерфейс головної сторінки

По центру – клікабельна карусель, натиснувши на яку відбудеться редирект на сторінку із відповідними до картинки товарами. Також видніється перший слайдер з новими товарами з високою знижкою.

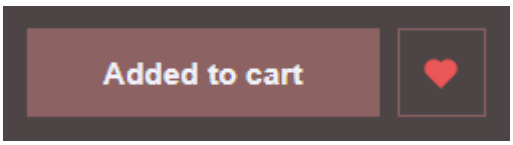


Рис. 3.2. Інтерактивні кнопки на картці продукта

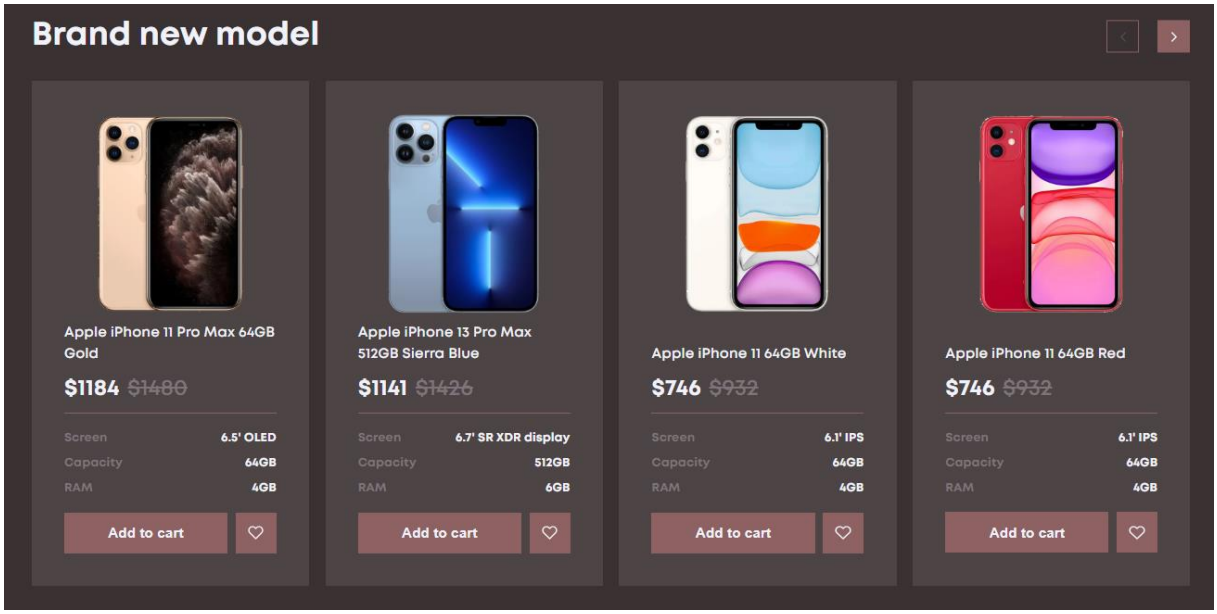


Рис. 3.3. Слайдер з новими товарами та знижкою

Ще один спосіб потрапити до каталогу продуктів – блок із категоріями.

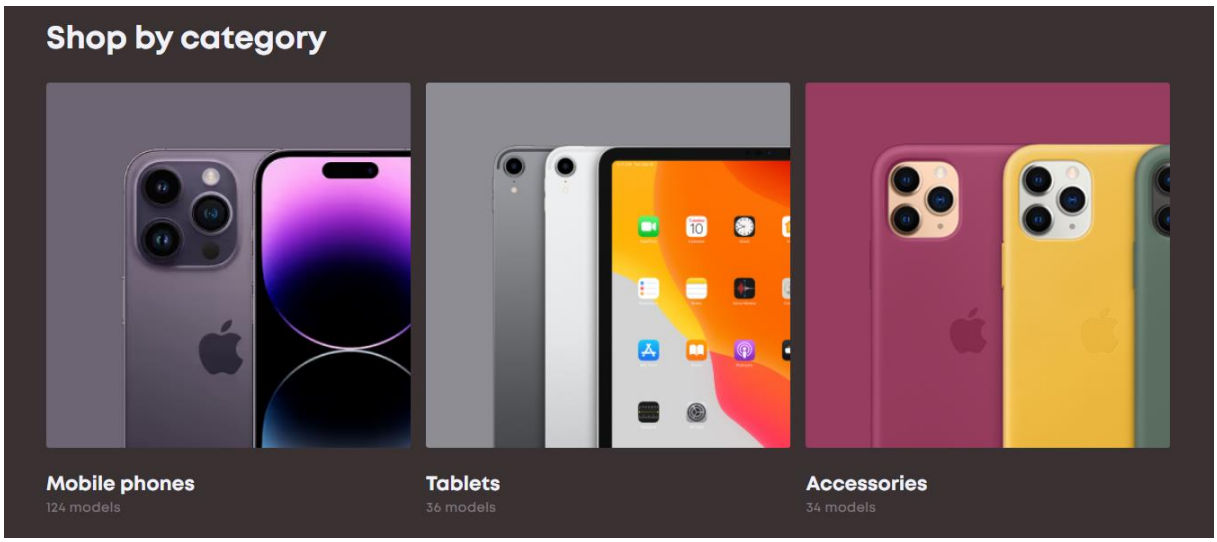


Рис. 3.4. Категорії гаджетів

Знизу розташований останній слайдер із техніки зі знижками та Footer, що слугує місцем для посилань на контакти і копірайт. Присутня кнопка Back to top, що повертає користувача наверх сторінки.

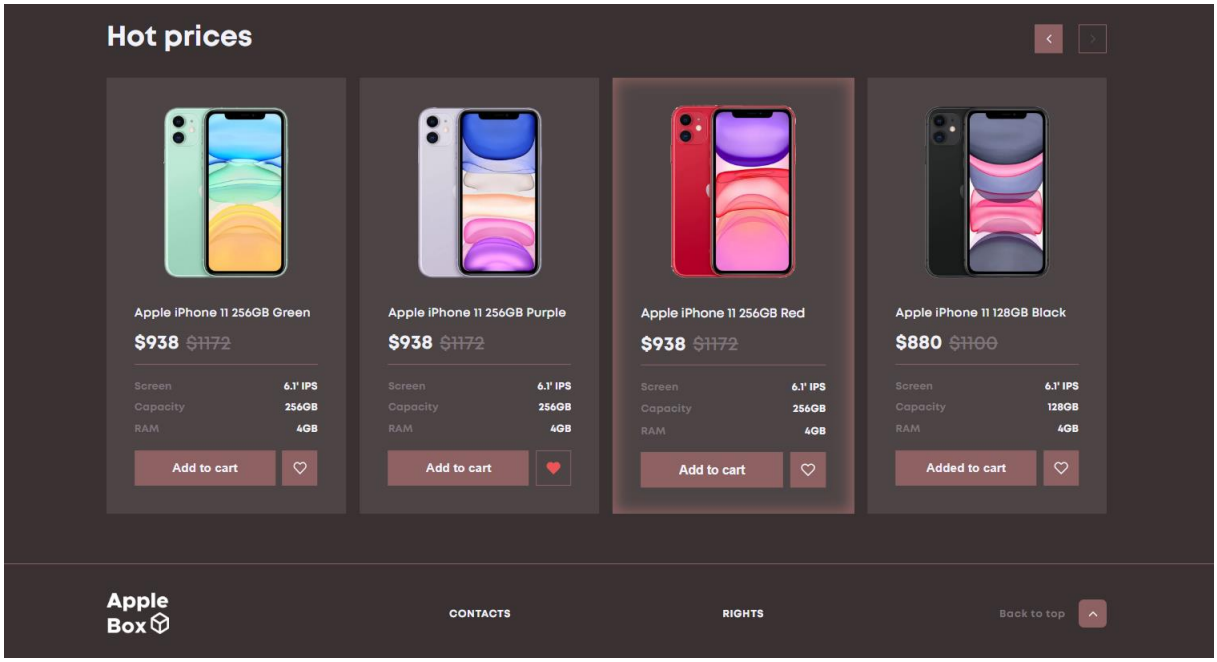


Рис. 3.5. Footer

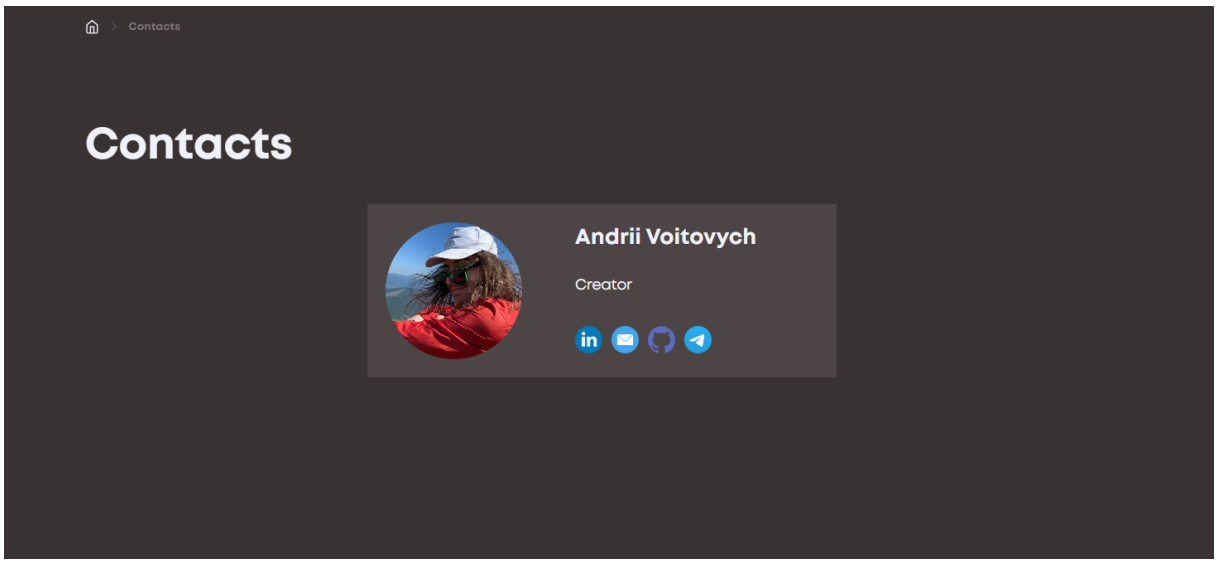


Рис. 3.6. Сторінка з контактами

Можливість зв’язатися за поштою і телеграмом. Також є змога відвідати LinkedIn або GitHub розробника.

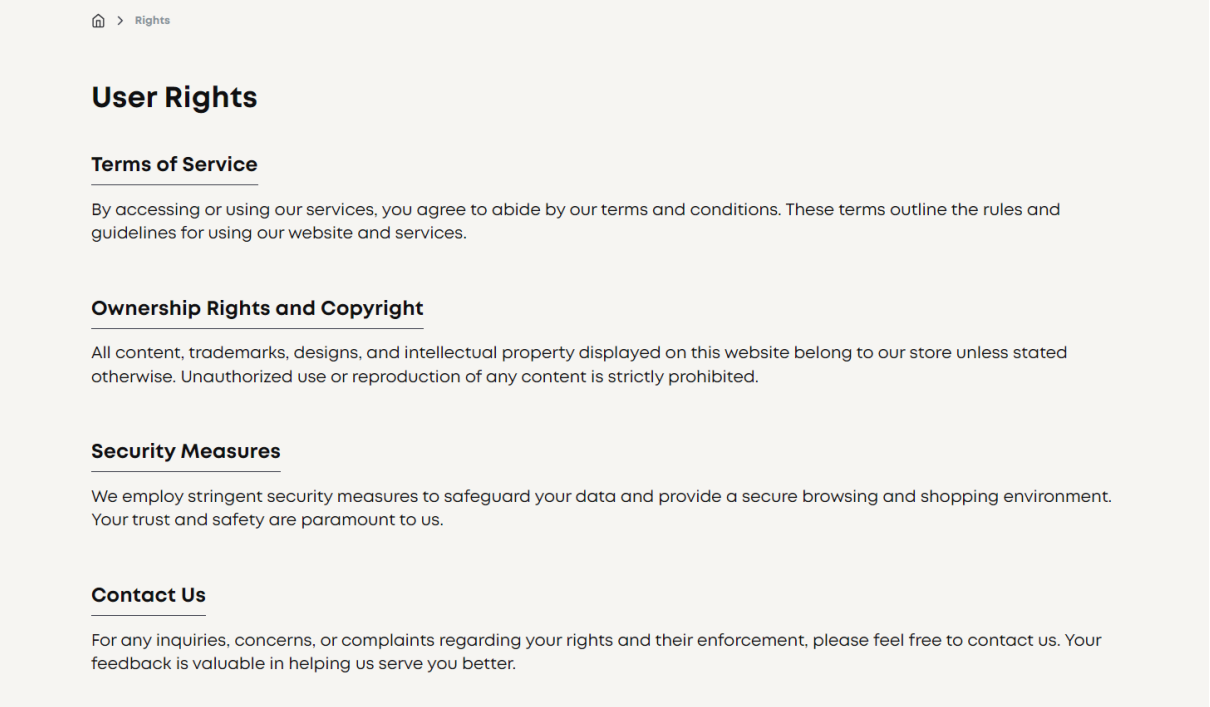


Рис. 3.6. Сторінка з правами користувача

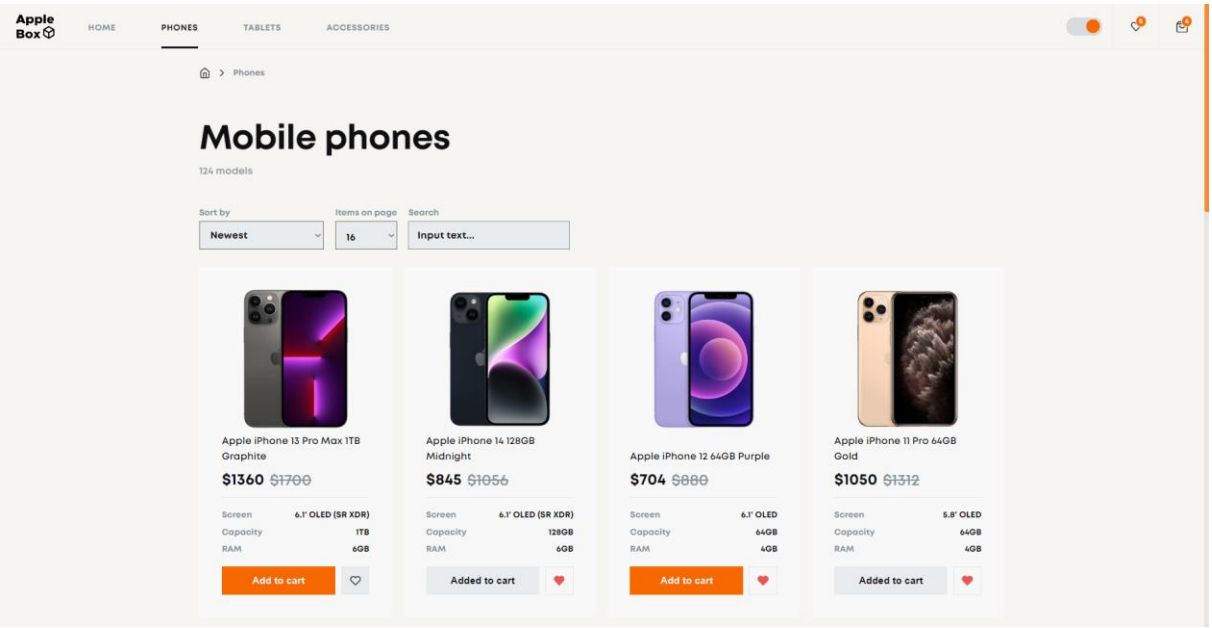


Рис. 3.7. Сторінка каталогу. Демонстрація світлої теми застосунку

Користувач має змогу відсортувати товари за бажаним алгоритмом, вибрати кількість відображення гаджетів на сторінці або скористатися пошуком.

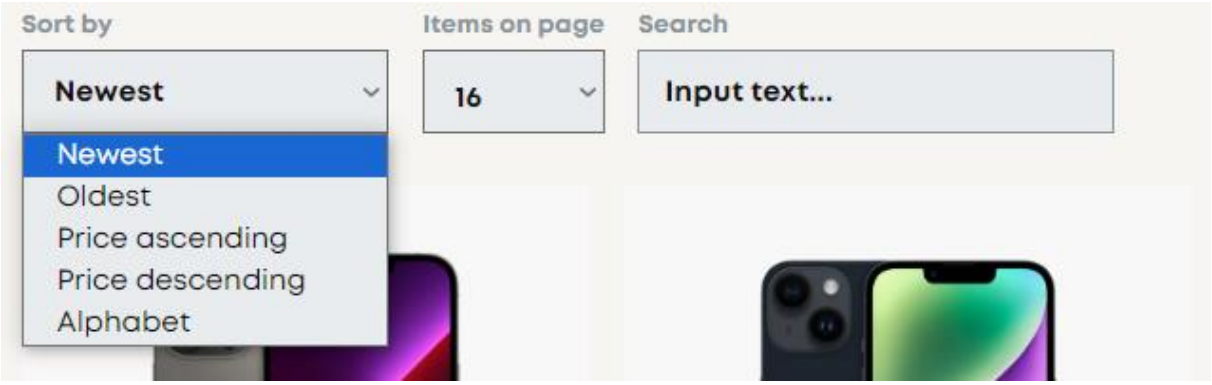


Рис. 3.8. Приклади сортування продуктів

Внизу сторінки знаходиться пагінація, яка перегортає сторінку товарів та піднімає користувача наверх сайту для зручності.

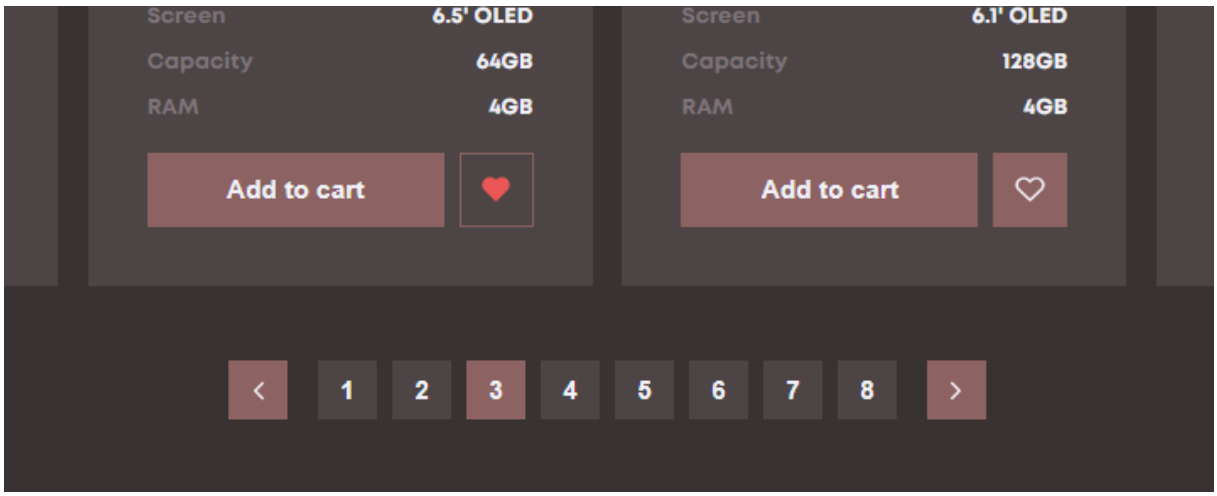
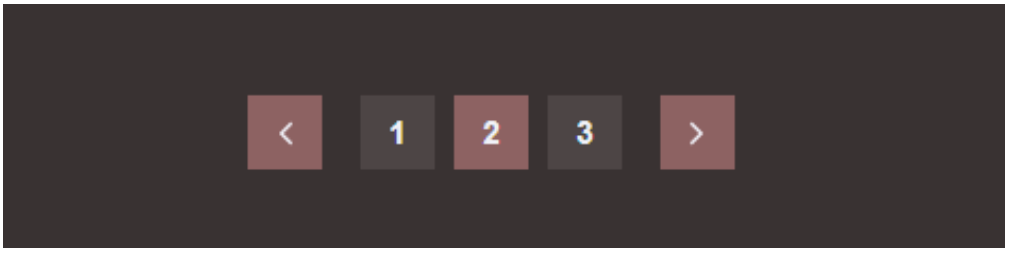


Рис. 3.9. Пагінація

Натиснувши на картку товару, користувач попадає на сторінку відповідного гаджета, де можна детальніше ознайомитись з його описом, характеристиками та наявністю кольорів. Кількість сторінок змінюється в залежності від кількості девайсів на сторінці.



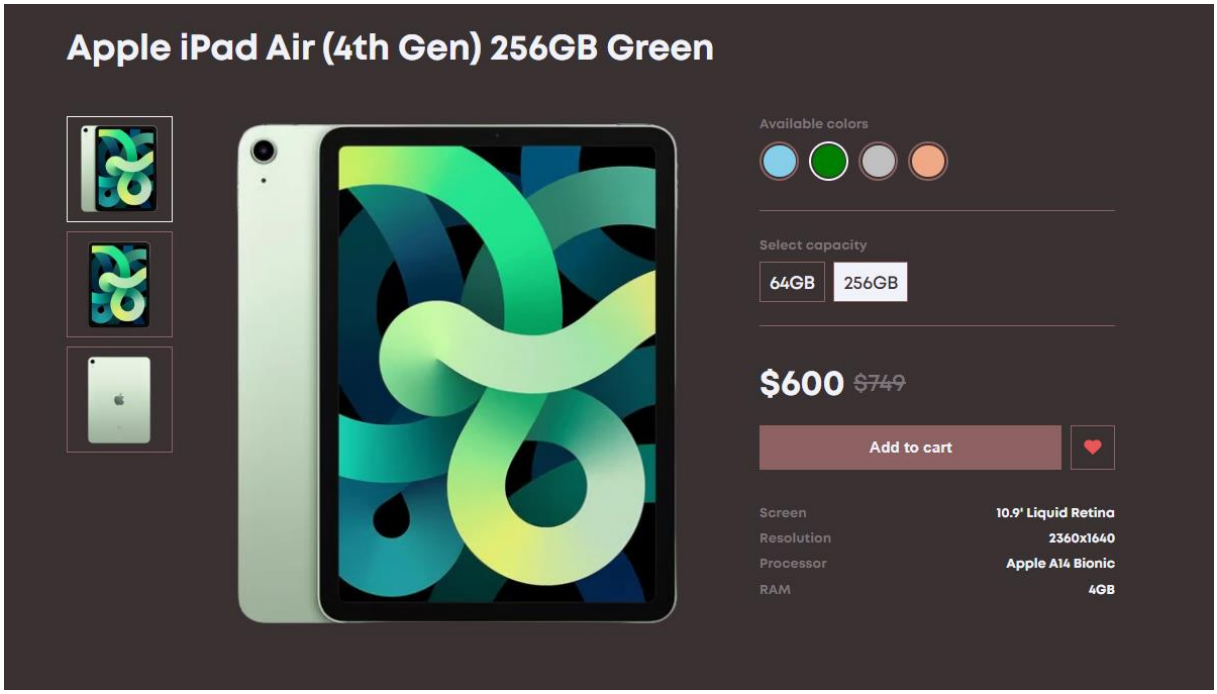


Рис. 3.10. Сторінка конкретного товару

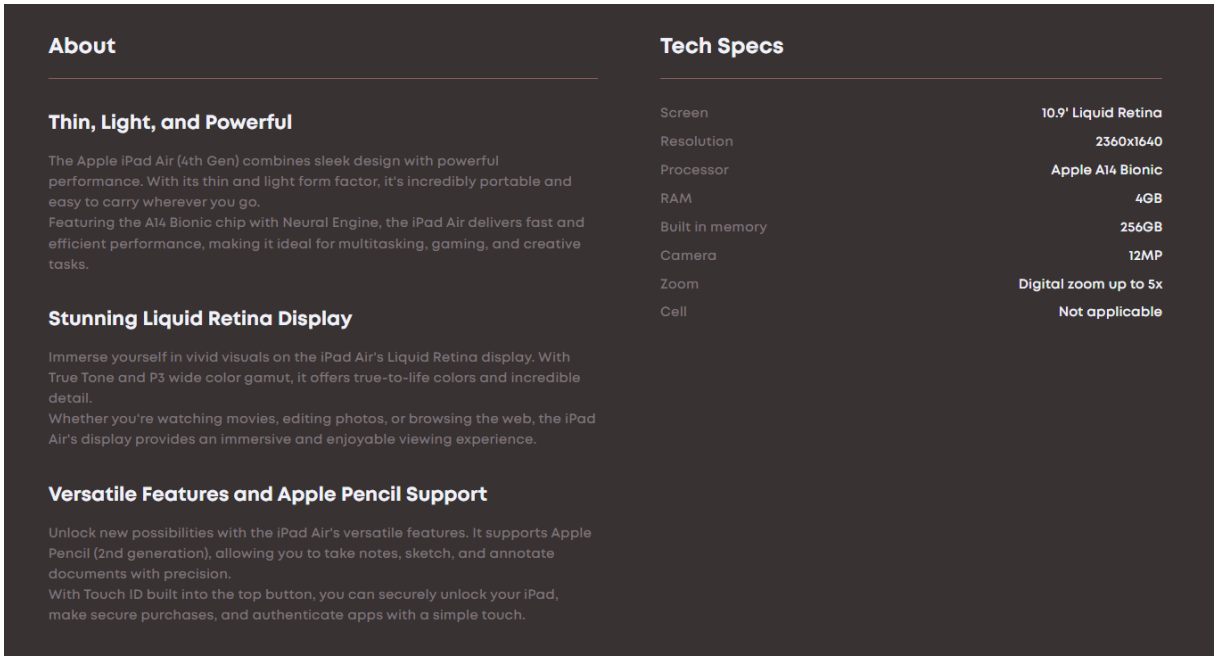


Рис. 3.11. Детальний опис товару

Якщо натиснути на кнопку із серцем – продукт потрапляє до сторінки Favourites, де користувач може обрати найкращий варіант серед вподобаних раніше пристроїв.

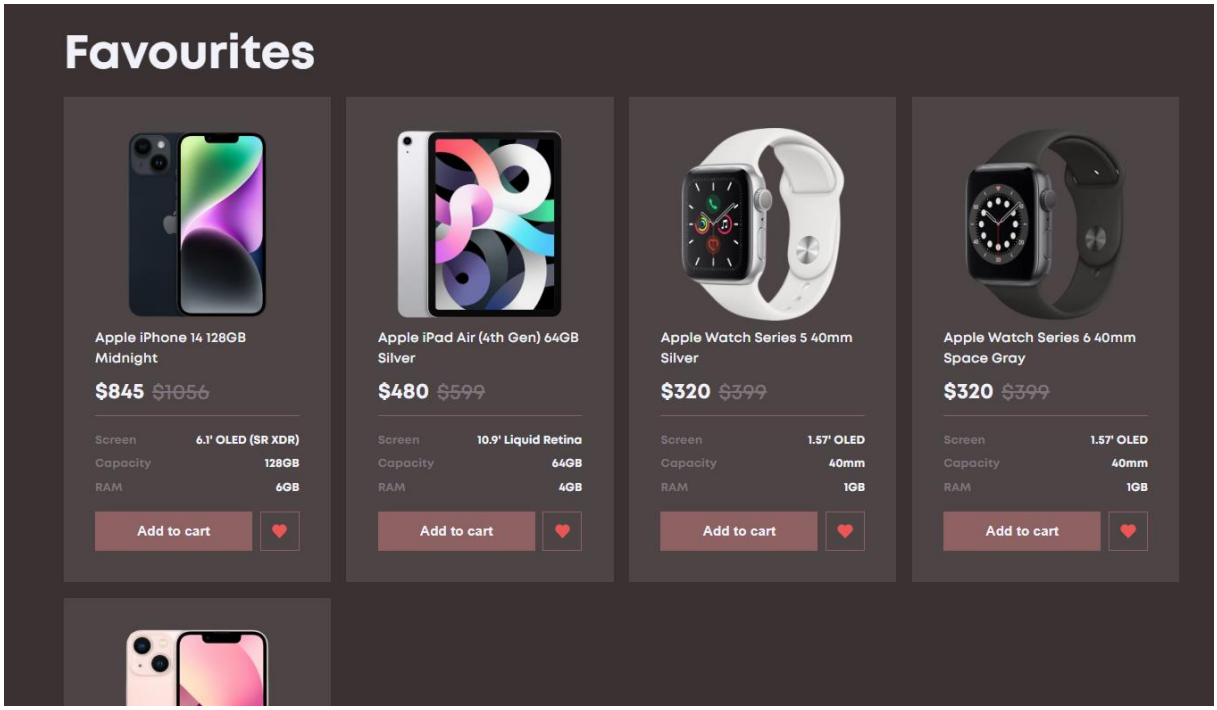


Рис. 3.12. Сторінка вподобаних гаджетів

У кошику є можливість вибрати бажану кількість девайсів для покупки. Окреме поле відображає загальну вартість товарів.

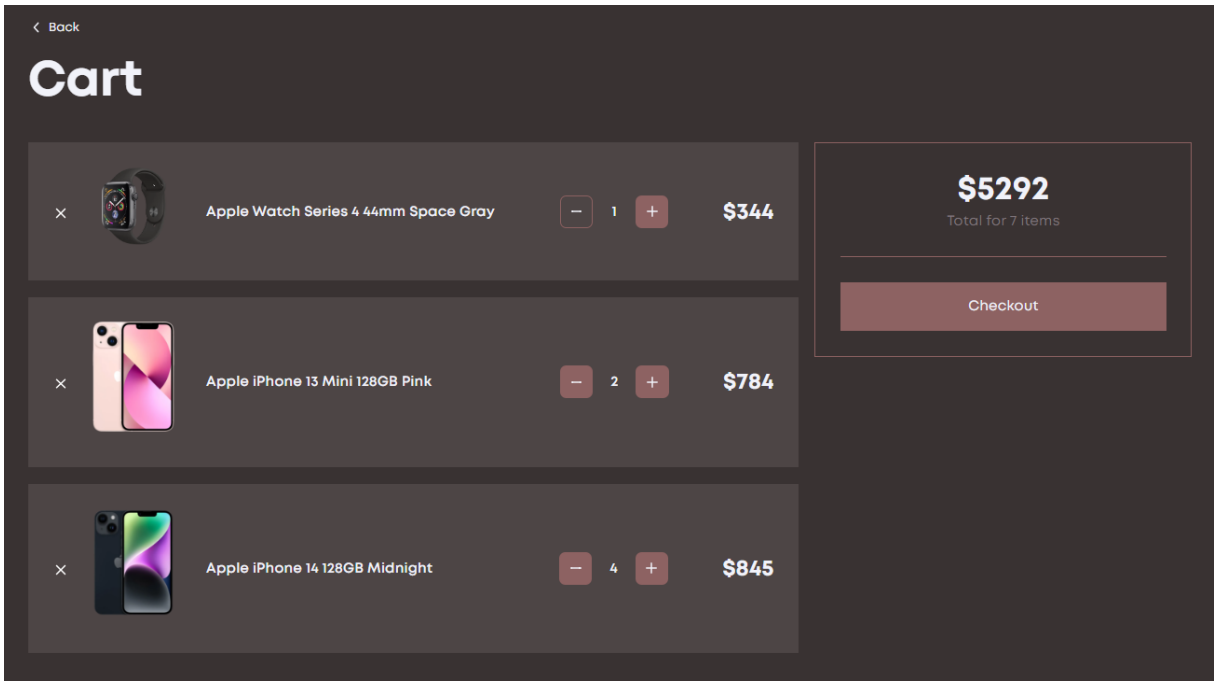


Рис. 3.13. Кошик

Кнопка Checkout відповідає викликає модальне вікно, яке вимагає підтвердження оплати користувачем.

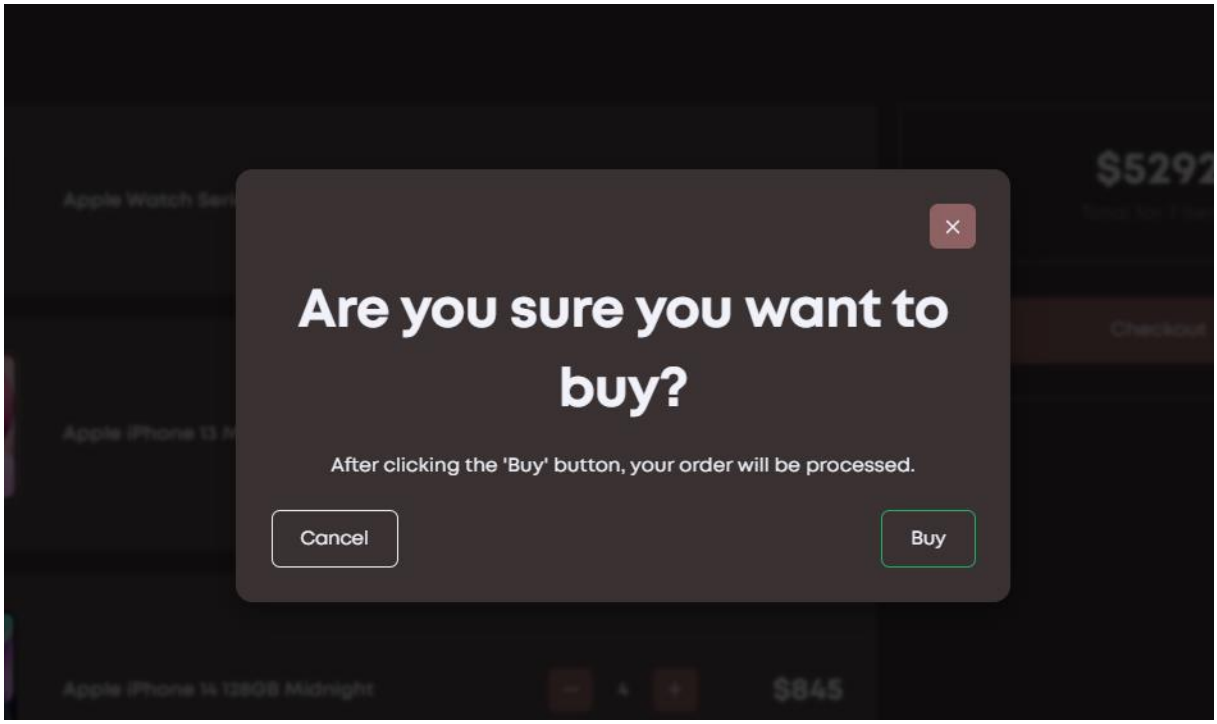


Рис. 3.14. Модальне вікно підтвердження покупки

ВИСНОВКИ

Під час виконання даної роботи було розроблено веб-додаток, який успішно реалізував функціональність електронної комерції. Проєкт охоплював усі основні етапи створення сучасного веб-додатку, починаючи від проєктування архітектури до впровадження. Основною метою було забезпечення користувачів зручним інтерфейсом та інтуїтивно зрозумілою навігацією, що вдалося досягти завдяки використанню сучасних технологій, таких як React, Redux та React Router.

Особливу увагу було приділено розробці надійної та масштабованої бекенд-системи, що надає швидкий та безпечний доступ до даних. Використання Node.js з Express.js та Sequelize для роботи з базою даних дозволило створити ефективну і гнучку серверну частину.

Крім того, розробка інтерфейсу користувача включала створення різноманітних компонентів, таких як кошик покупок, слайдер продуктів та модальні вікна. Усі ці компоненти були розроблені з урахуванням принципів модульності та повторного використання коду, що сприяє легкості підтримки та масштабування проєкту. Були враховані всі вимоги до доступності та адаптивності, що дозволило зробити додаток зручним для використання на різних пристроях.

Отже, можна відзначити, що всі поставлені задачі були успішно виконані, а отримані результати підтверджують високу якість та ефективність розробленого додатку. Проєкт продемонстрував застосування сучасних технологій та підходів у веб-розробці. Отримані знання та досвід будуть корисними в подальшій професійній діяльності та сприятимуть розвитку навичок у сфері інформаційних технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Д.Д. Гнатченко, Т.О. Гнатченко, В.А. Маляр. Застосування сучасних інформаційних технологій для організації торгівлі у мережі інтернет. Інформаційні технології в економіці. № 41. 2020. С. 163–166. URL: http://mdcs.knuba.edu.ua/article/view/203828/203576
2. Менів Андрій. Різновиди інтернет-магазинів [Електронний ресурс]. URL: https://pik.net.ua/2024/05/14/riznovydy-internet-magazyniv-vybir-najkrashhogo-dlya-vashogo-biznesu/
3. TypeScript Documentation [Електронний ресурс]. URL: https://www.typescriptlang.org/docs/
4. React Documentation [Електронний ресурс]. URL: https://uk.legacy.reactjs.org/docs/getting-started.html

5. What are the advantages of React.js? [Електронний ресурс]. 2023. URL: https://www.geeksforgeeks.org/what-are-the-advantages-of-react-js/

6. Sass Documentation [Електронний ресурс]. URL: https://sass-lang.com/documentation/

7. Feature Sliced Design Documentation [Електронний ресурс]. URL: https://feature-sliced.design/docs/get-started/overview

8. Model View Controller Docs [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Glossary/MVC

9. PostgreSQL Documentation [Електронний ресурс]. URL: https://www.postgresql.org/docs/current/intro-whatis.html

Додаток А

Код компонента CategoryProducts.

```
import React, { ChangeEvent, useContext, useEffect, useMemo, useState }
from "react";
import { useLocation, useSearchParams } from "react-router-dom";
import { useAsyncValue } from "react-router";
import { ItemCard } from "../../../entities/ItemCard/ItemCard";
import styles from "../CatalogPage.module.scss";
import { ProductResponse } from "../../../types/ProductResponse";
import { PageButton } from "../../../shared/ui/PageButton/PageButton";
import { PageChangeButton } from
"../../../shared/ui/PageChangeButton/PageChangeButton";
import { getNumbers } from "../../../shared/utils/getNumbers";
import { ItemCardState } from
"../../../entities/ItemCardState/ItemCardState";
import crossIcon_light from "../../../static/Catalog/cross_light.svg";
import crossIcon from "../../../static/Catalog/cross.svg";
import { ThemeContext } from "../../../app/providers/ThemeProvider";

const limits = [16, 24, 48];
const sorts = [
  { name: "Newest", value: "time" },
  { name: "Oldest", value: "time-desc" },
  { name: "Price ascending", value: "price" },
  { name: "Price descending", value: "price-desc" },
  { name: "Alphabet", value: "name" }
];

const arrayRange = (start: number, stop: number, step: number) =>
  Array.from(
    { length: (stop - start) / step + 1 },
    (value, index) => start + index * step
  );

export const CategoryProducts: React.FC<{ state?: "loading" | "error" }>
= ({ state }) => {
  const location = useLocation();
  const category = location.pathname.slice(1);
  const { theme } = useContext(ThemeContext);
```

```
  const data = useAsyncValue() as ProductResponse;
  const [searchParams, setSearchParams] = useSearchParams();

  const params = useMemo(() => {
    return {
      page: searchParams.get("page") || "",
      limit: searchParams.get("limit") || "16",
      query: searchParams.get("query") || "",
      sortBy: searchParams.get("sortBy") || "",
      desc: searchParams.get("desc") || ""
    };
  }, [searchParams, category]);

  const pages = (state) ? [] : getNumbers(0, (data.total /
+params.limit));

  const [currentPage, setCurrentPage] = useState(params.page);
  const [limit, setLimit] = useState(params.limit);
  const [sort, setSort] = useState({ type: params.sortBy, isDesc:
!!params.desc });
  const [query, setQuery] = useState(params.query);

  const stateArray = arrayRange(1, +limit, 1);

  useEffect(() => {
    setSearchParams(params => {
      params.set("page", currentPage.toString());
      params.set("limit", limit.toString());
      if (query) {
        params.set("query", query?.toString());
      }
      if (sort.type) {
        params.set("sortBy", sort.type);
        params.set("desc", sort.isDesc.toString());
      }
      return params;
    });
  }, [currentPage, limit, query, sort]);

  useEffect(() => {
    setCurrentPage(() => "0");
    setLimit(() => params.limit);
```

```
    setSort({ type: "time", isDesc: false });
    setQuery(() => "");
  }, [category]);

  const handleLimitChange = (value: string) => {
    setCurrentPage("0");
    setLimit(value);
  };

  const handleSortChange = (event: React.ChangeEvent<HTMLSelectElement>)
=> {
    const [type, order = "asc"] = event.target.value.split("-");
    setSort({
      type,
      isDesc: order === "desc"
    });
  };

  const handleQueryChange = (event: ChangeEvent<HTMLInputElement>) => {
    if (!event.target.value) {
      setQuery("");
      searchParams.delete("query");
    } else {
      setQuery(event.target.value);
    }
  };

  const handleCrossClick = () => {
    setQuery("");
    searchParams.delete("query");
  };

  return (
    <>
      {category === "phones" && <h1 className={styles.title}>Mobile
phones</h1>}
      {category === "tablets" && <h1
className={styles.title}>Tablets</h1>}
      {category === "accessories" && <h1
className={styles.title}>Accessories</h1>}

      <p className={styles.modelCount}>
```

```
        {state
          ? (state === "loading"
            ? "Loading models"
            : "Could not load models"
          )
          : `${data.total} models`
        }
      </p>

      <div className={styles.selectors}>
        <div className={styles.selectors__Wrapper}>
          <label htmlFor="Sort by"
className={styles.selectors__Label}>Sort by</label>

          <select
            name="Sort by"
            id="Sort"
            className={styles.selectors__Sort}
            onChange={handleSortChange}
          >
            {sorts.map(sort =>
              <option
                key={sort.value}
                className={styles.selectors__Option}
                value={sort.value}
                selected={params.sortBy === sort.value}
              >
                {sort.name}
              </option>
            )}
          </select>
        </div>

        <div className={styles.selectors__Wrapper}>
          <label htmlFor="Items on page"
className={styles.selectors__Label}>Items on page</label>

          <select
            name="Items on page"
            id="Items"
            className={styles.selectors__Items}
            onChange={event => handleLimitChange(event.target.value)}
```

```
          >
            {limits.map(number =>
              <option
                className={styles.selectors__Option}
                key={number}
                value={number}
                selected={number === +limit}
              >
                {number}
              </option>)}
          </select>
        </div>

        <div className={styles.selectors__Wrapper}>
          <label htmlFor="searchBar"
className={styles.selectors__Label}>Search</label>

          <div className={styles.selectors__Search}>
            <input
              className={styles.selectors__Search__Input}
              type="text"
              onChange={handleQueryChange}
              value={query}
              placeholder="Input text..."
            />
            {query && <button
              type="button"
              className={styles.selectors__Search__Button}
              onClick={handleCrossClick}
            >
              <img src={theme === "light" ? crossIcon_light : crossIcon}
alt="delete" />
            </button>}
          </div>
        </div>
      </div>
      <div className={styles.products}>
        {(!state && data.total === 0) &&
          <h2 className={styles.products__nothing_found}>
            Nothing found by query {`${params.query}`}
          </h2>
        }
```

```
        {state
          ? stateArray.map(id => (
            <ItemCardState key={id} state={state} />
          ))
          : data.data.map(item => (
            <ItemCard key={item.id} phone={item} />
          ))
        }
      </div>

      {pages.length > 1
        && (
          <div className={styles.pagination}>
            <PageChangeButton
              direction="prev"
              selected={+currentPage}
              onPageChange={setCurrentPage}
            />
            <div className={styles.pagination__Pages}>
              {pages.map(pageNumber => (
                <PageButton
                  key={pageNumber}
                  pageNumber={pageNumber}
                  selected={+currentPage}
                  onPageChange={setCurrentPage}
                />
              ))}
            </div>
            <PageChangeButton
              direction="next"
              selected={+currentPage}
              onPageChange={setCurrentPage}
              maxPage={pages.length.toString()}
            />
          </div>
        )
      }
    </>
  );
};
```