МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
_____
(підпис)
грудня 202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Інформаційне та програмне забезпечення адаптивного веб-сайту для продажу взуття»

здобувача групи ІН-02 Білецького Дмитра Олексійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Дмитро Білецький
(підпис)

Керівник, доцент комп'ютерних наук, доцент, к.т.н. Валентина Боровик _____
(підпис)

Суми – 2024

Сумський державний університет

Центр заочної, дистанційної та вечірньої форм навчання

Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-02 Білецького Дмитра Олексійовича

1. Тема роботи: «Інформаційне та програмне забезпечення адаптивного веб-сайту для продажу взуття»
затверджую наказом по СумДУ від *«22» квітня 2024 р. № 0414-VI*

2. Термін здачі здобувачем кваліфікаційної роботи *до 01 червня 2024 року*

3. Вхідні дані до кваліфікаційної роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити):

1) *Інформаційний огляд, аналіз предметної області, постановка задачі, моделювання інформаційної системи.*
2) *Вибір методів рішення задачі.*
3) *Програмна реалізація.*
4) *Взаємодія користувача з системою.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ (підпис)

Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Затвердження теми роботи*	22.04.2024	
2	*Вивчення та аналіз задачі*	7.05.2024	
3	*Поглиблене дослідження бібліотек, інструментів, технологій, що будуть використовуватися*	10.05.2024	
4	*Програмна реалізація системи*	19.05.2024	
5	*Аналіз отриманих результатів*	22.05.2024	
6	*Оформлення пояснювальної записки до кваліфікаційної роботи*	25.05.2024	

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 129 стр., 104 рис., 2 таблиці, 22 додатки, 23 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена створенню адаптивного веб-сайту для продажу взуття, враховуючи аспект зростання інтернет-торгівлі, зростання користування мобільними пристроями, зростання вимог користувачів в цілому.

Об’єкт дослідження — функціональні аспекти створення адаптивного веб-сайту, його інтерфейс, дизайнерські та функціональні компоненти.

Мета роботи — розробка адаптивного веб-сайту, який використовуватиметься у сфері продажу взуття.

Методи дослідження — аналіз сучасних тенденцій в електронній комерції, аналіз користувацького досвіду, порівняльний аналіз з іншими ринковими учасниками.

Результати — розроблено front-end частину веб-сайту для продажу взуття та з’єднано її зі стороннім бекендом, при цьому забезпечено функціонал розбиття товарів на категорії, їх пошук в межах інтернет-магазину, сортування за різними параметрами. Забезпечено пагінацію та відображення окремої сторінки кожного товару. Реалізовано функціонал кошику.

ІНФОРМАЦІЙНА СИСТЕМА, АДАПТИВНИЙ ВЕБ-САЙТ, WEB-РОЗРОБКА, FRONT-END, BACK-END, REACT, JAVASCRIPT, REDUX, AXIOS, SASS, MOCKAPI, VERCEL

ЗМІСТ

ВСТУП

У сучасному цифровому світі інтернет-торгівля стає все більш популярною і затребуваною формою комерційної діяльності. За допомогою інтернет-магазинів, покупці можуть легко придбати товари та послуги, обираючи з широкого асортименту продукції. Електронні магазини не зможуть замінити традиційні магазини, але обов'язково розширять ринок та сферу застосування.

Зростаюче різноманіття технологій та рівень очікувань споживачів вимагають постійного вдосконалення веб-ресурсів для забезпечення зручності та ефективності взаємодії з клієнтами.

Адаптивні веб-сайти стають ключовим інструментом в цьому процесі, забезпечуючи користувачам оптимальний досвід перегляду, незалежно від пристрою, на якому вони звертаються до сайту.

Актуальність розробки веб-сайту пояснюється наступними факторами:

- Зростання інтернет-торгівлі. Інтернет-торгівля продовжує зростати з кожним роком, привертаючи увагу як великих корпорацій, так і малих підприємств. Розробка веб-додатку дозволяє бізнесам отримати доступ до широкої аудиторії клієнтів через Інтернет та розширити свої можливості продажу.
- Зростання користування мобільними пристроями. Розробка адаптивного веб-додатку забезпечує оптимальний доступ користувача на різних пристроях, що дозволяє бізнесам збільшити свою доступність та залучити більше клієнтів.
- Зростання конкуренції. Розробка веб-додатку дозволяє бізнесам відрізнятися від конкурентів, надаючи клієнтам зручний, ефективний та привабливий досвід покупок.
- Зростання вимог споживачів. Сучасні споживачі вимагають зручності, швидкості та персоналізації від своїх інтернет-покупок. Розробка веб-додатку дозволяє бізнесам відповісти на ці вимоги, надаючи

користувачам зручну навігацію, швидкий пошук товарів та персоналізовані рекомендації.

Метою роботи є розробка адаптивного веб-сайту, який використовуватиметься у сфері продажу взуття.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1. Аналіз предметної області

Коли йде мова про успішний інтернет-магазин з точки зору функціональних можливостей, існують деякі ключові критерії, які визначають зручність та ефективність взаємодії користувачів з веб-сайтом:

• Дизайн. Візуальний аспект веб-сайту має безпосередній вплив на перше враження користувача і його подальше рішення щодо продовження взаємодії з додатком. Важливо, щоб інтерфейс був привабливий, мав правильний підбір кольорів, шрифтів, зображень. Крім того, дизайн повинен бути адаптивним, тобто коректно відображатися на різних типах пристроїв, включаючи комп'ютери, планшети та мобільні телефони. 66% користувачів мережі Інтернет (рис. 1.1), для доступу використовують саме мобільні телефони або смартфони [1], тому оптимізація веб-сайту для мобільних пристроїв стає невід'ємною складовою успіху.

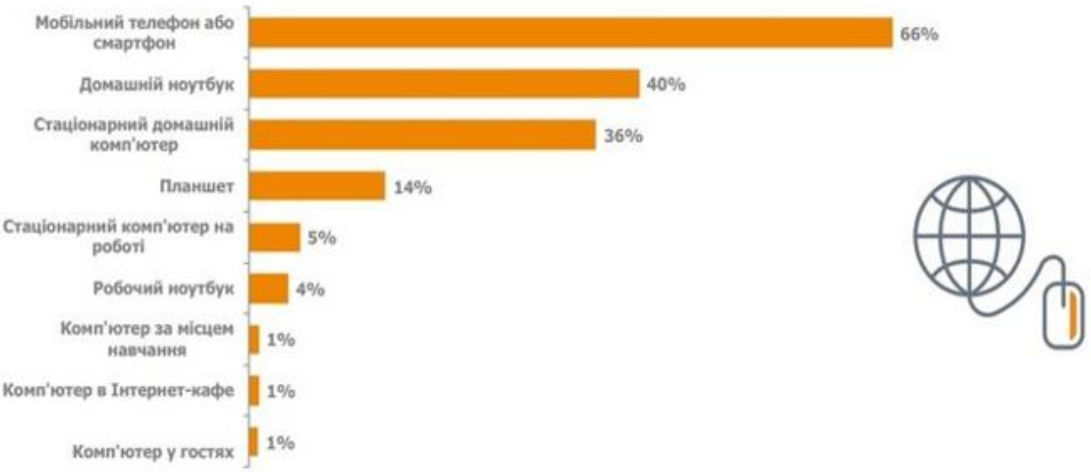


Рисунок 1.1 – Статистичні дані щодо використання пристроїв для доступу до Інтернету [1]

• Пошук та сортування. Користувачі очікують знайти потрібний їм товар швидко та без зайвих зусиль. Для цього необхідно мати систему пошуку, яка дозволяє точно визначити потрібний товар. Крім того, можливість сортування товарів за різними параметрами дозволяє користувачам легко відібрати товари відповідно до їхніх потреб та вимог.

• Категорії і навігація. Інтуїтивно зрозуміла та легка у використанні навігація по категоріях також важлива, оскільки вона допомагає користувачам швидко зорієнтуватися в асортименті товарів та здійснити вибір. При цьому, проста та логічна структура категорій дозволяє зберегти час та зусилля покупців.

• Пагінація. Розміщення контенту на декількох сторінках в межах головної сторінки з використанням пагінації забезпечує зручний доступ до великого обсягу товарів без перевантаження однієї сторінки.

Проаналізуємо декілька інтернет-магазинів зі схожою тематикою і функціоналом.

Першим сайтом для аналізу є https://loveandlive.ua [2]

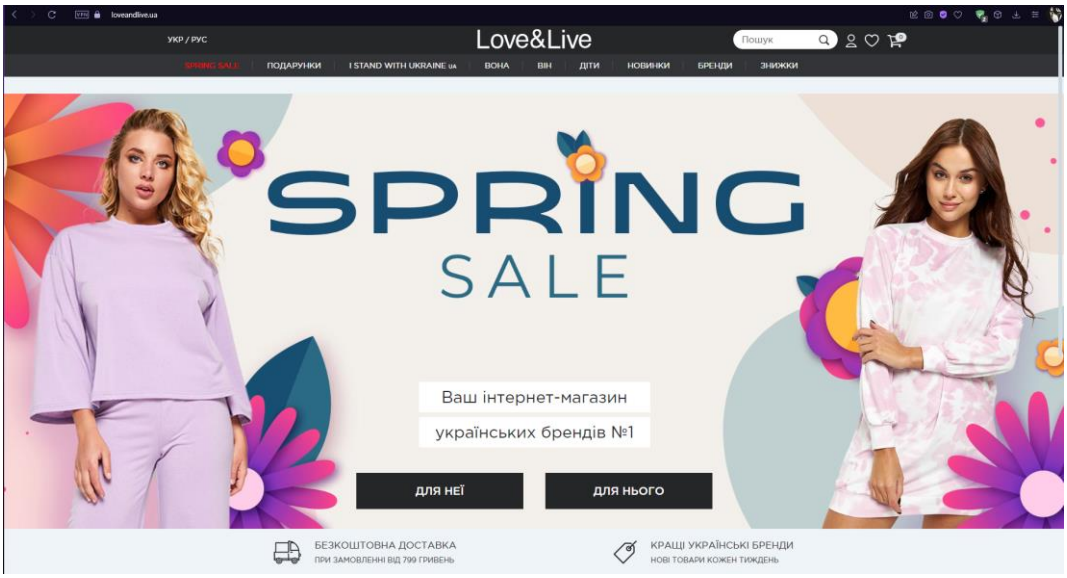


Рисунок 1.2 – Вигляд головної сторінки інтернет-магазину Love&Live

Першим, що спадає до уваги – дизайн. Слід зауважити, що, розташування компонентів веб-сайту не сприяє кращому користувацькому досвіду, така розмітка відволікає користувача, складно сфокусуватися при взаємодії з таким інтернет-магазином. До того ж, варто було б провести роботу з підбору шрифтів, оскільки даний варіант не є привабливим.

Крім того, на сайті є проблеми з адаптивною верстkою (рис. 1.1.3). Як можна побачити, на роздільній здатності смартфону Apple iPhone SE (мінімально можлива роздільна здатність екрану, що використовують користувачі) маємо проблему з відображенням навігаційної панелі, блок з назвою сайту і полем

пошуку перекриває її. До того ж, кнопка переходу до кошика виходить за межі екрану. Оскільки, зображення товарів не можуть вміститися в межах екрану, було б доцільно зменшити їх розмір.

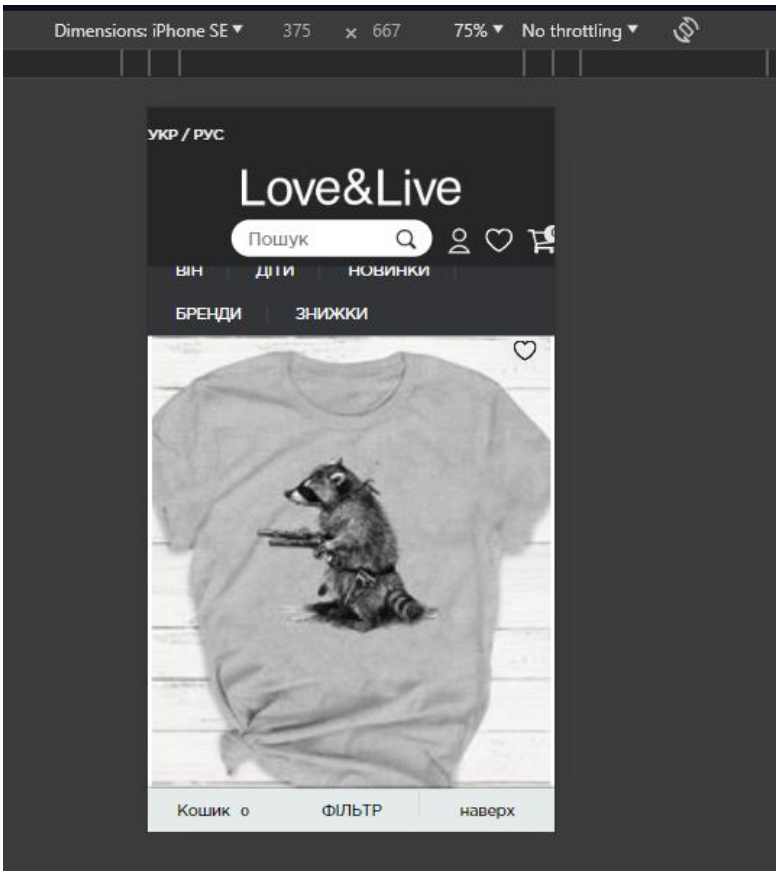


Рисунок 1.3 – Вигляд адаптивної верстки інтернет-магазину Love&Live

Однак, на сайті реалізовано функціонал пошуку, сортування і фільтрації товарів. Самі товари розбиті на категорії і забезпечено можливість навігації між ними. Крім того, реалізовано пагінацію, яка дозволяє демонструвати товари на декількох сторінках, а не в межах одної нескінченної. Все вищеперераховане є корисними доповненнями, які забезпечать зручність і кращий користувацький досвід в цілому.

Наступним сайтом розглянемо https://assorti-odessa.com.ua/ua/ [3]

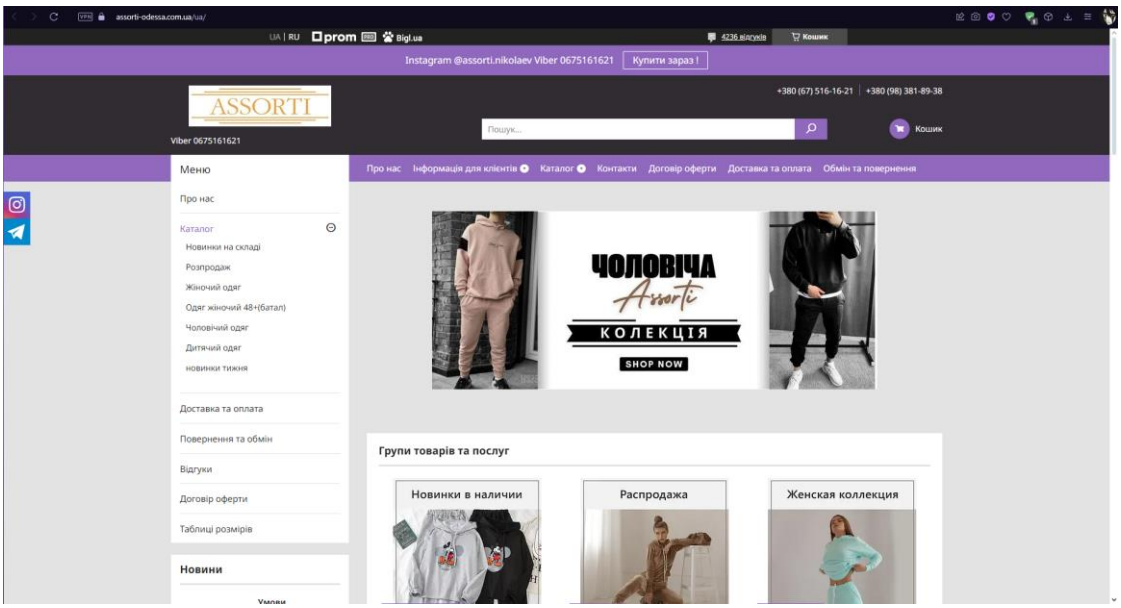


Рисунок 1.4 – Вигляд головної сторінки інтернет-магазину ASSORTI

Він має приємний дизайн, і так само як і в інтернет-магазині Love&Live, у ньому реалізовано розбиття товарів на категорії, можливість пошуку цих товарів і їх фільтрування, пагінація, однак сортування реалізовано не повністю, є можливість відібрати товари лише за ціною. Бракує можливості сортувати товари за назвою і популярністю, це б забезпечило кращий досвід від взаємодії з інтернет-магазином. Крім того, веб-сайт має проблеми з локалізацією. На головній сторінці назви груп товарів наведено російською мовою, в той час як весь інший контент представлено українською мовою.

Також був проаналізований веб-сайт https://www.cropp.com/ua/uk/ [4]

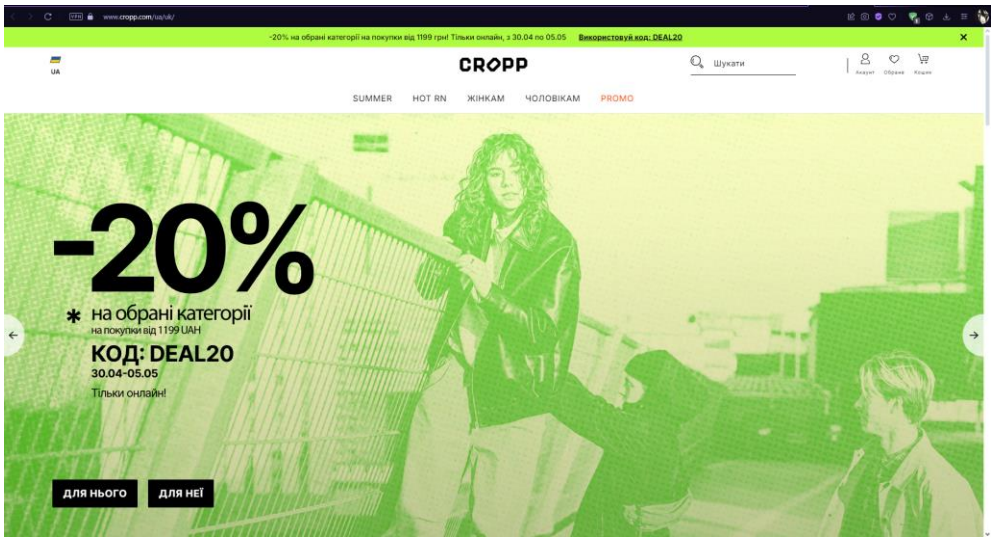


Рисунок 1.5 – Вигляд головної сторінки інтернет-магазину CROPP

Він має приємний для демонстрації та взаємодії інтерфейс, на ньому реалізовано функціонал пошуку, фільтрування, сортування товарів, товари розбиті на категорії і забезпечено можливість навігації між ними. Однак, інтернет-магазину бракує пагінації, через це категорії, що мають сотні товарів, користувач вимушений скролити нескінченно вниз для того щоб переглянути всі товари, що його цікавлять. Демонстрація товарів на декількох сторінках покращила б користувацький досвід.

Підсумовуючи, в таблиці 1.1 наведено загальний аналіз обраних веб-сайтів:

Таблиця 1.1 Аналіз критеріїв обраних веб-сайтів

Критерій	Love&Live	ASSORTI	CROPP
Приємний дизайн	-	+	+
Функціонал пошуку товарів	+	+	+
Функціонал фільтрування товарів	+	+	+
Функціонал сортування товарів	+	-	+
Функціонал навігації між котегоріями	+	+	+
Функціонал пагінації	+	+	-
Забезпечення адаптивності	-	+	+

Виконавши аналіз інтернет-магазинів з подібним призначенням, було вирішено розробити веб-додаток, який не мав би недоліків вищенаведених веб-сайтів. З урахуванням того, що щодня, в усьому світі, зокрема в Україні все більшого поширення набуває Інтернет торгівля та різні методи ведення бізнесу в Інтернеті [5], розробка та впровадження адаптивного інтернет-магазину стає актуальним завданням. Важливо забезпечити оптимальне середовище для користувачів веб-додатку, яке б відповідало їхнім потребам та очікуванням у зручності.

1.2. Постановка задачі

За результатами інформаційного огляду та аналізу аналогічних програмних продуктів, сформовано завдання даної роботи – створити веб-сайт для продажу взуття, реалізувавши при цьому наступне:

- створити модель інформаційної системи, а саме структуру сторінко веб-сайту та їх компонентів;
- забезпечити розбиття товарів на категорії та зручну навігацію між ними;
- забезпечити пагінацію;
- створити функціонал пошуку товарів;
- створити функціонал сортування товарів (за назвою, за ціною, за популярністю – за спаданням та зростанням);
- забезпечити відображення окремої сторінки з детальним описом для кожного товару;
- створити функціонал кошику замовлень;
- забезпечити адаптивну верстку;
- розгорнути веб-сайт на хостингу.

1.3. Моделювання інформаційної системи

Розглянемо структуру сторінок веб-сайту, що реалізується, та їх компоненти (рис. 1.6):

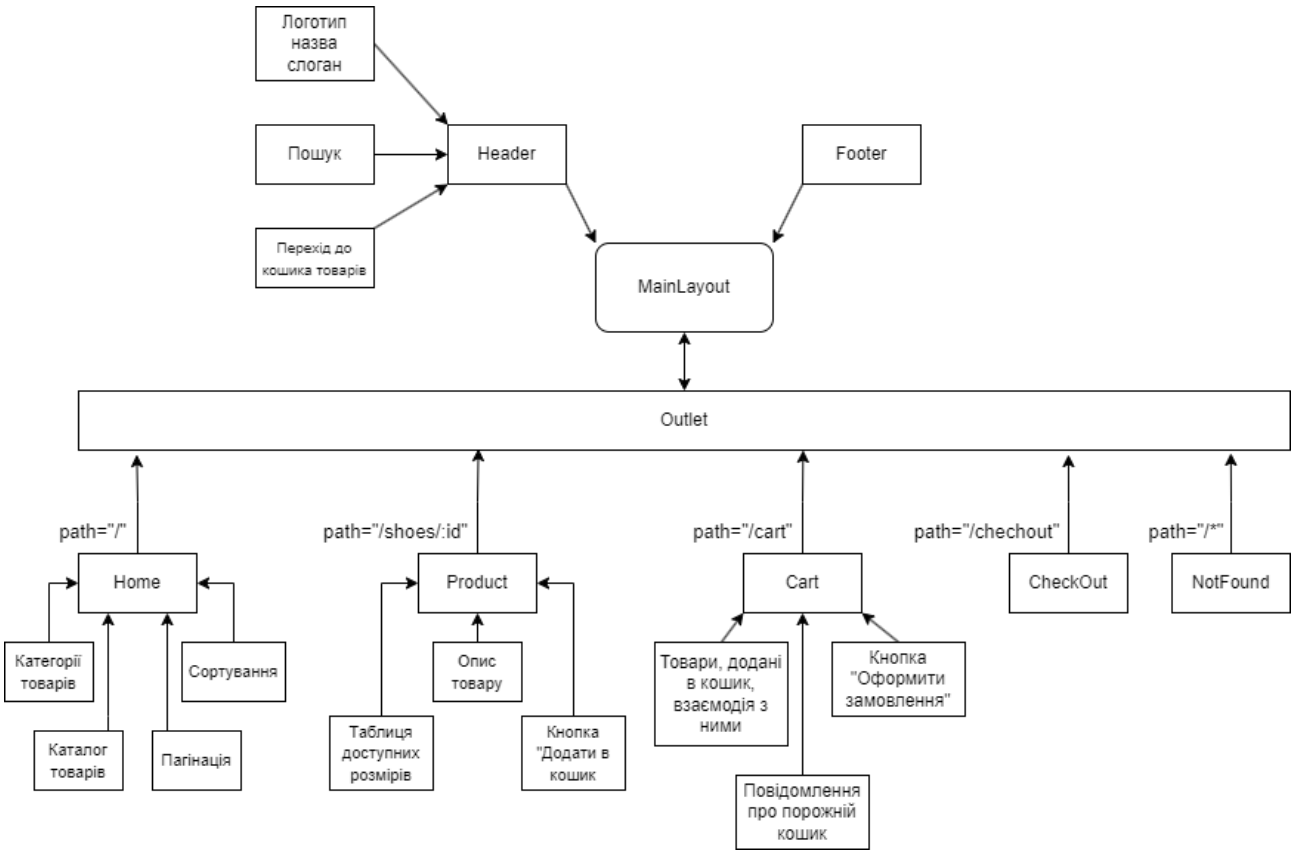


Рисунок 1.6 – Структура сторінок веб-сайту та їх компоненти

Наповнення сторінок визначає компонент MainLayout, він обов’язково включає в себе хедер сайту, що містить компоненти: логотип сайту, його назву, слоган; пошуку товарів; кнопку переходу до кошика товарів. Основний вміст сторінки визначає компонент Outlet, який в залежності від url адреси буде відображати наступні сторінки:

- path = “/” – означає, що користувач знаходиться на домашній сторінці веб-сайту. Вона містить наступні компоненти: категорії товарів; сортування; каталог товарів; пагінація.
- path = “/shoes/:id” – означає, що користувач знаходиться на сторінці товару, де :id – динамічний параметр, що визначає конкретний товар. Сторінка містить наступні компоненти: опис товару; таблиця доступних розмірів взуття; кнопка «Додати в кошик».
- path = “/cart” – означає, що користувач знаходиться на сторінці кошику. Вона містить наступні компоненти: товари, що додані в кошик, з врахуванням обраних розмірів; кнопка «Оформити замовлення»; у

випадку, якщо до кошика не додано жодного товару – повідомлення про порожній кошик

- path = “/checkout” – означає, що користувач знаходиться на сторінці оформлення замовлення. Вона містить інформаційне повідомлення.
- path = “/*” – означає, що користувач перейшов за url-адресою неіснуючої сторінки. Компонент NotFound містить в собі відповідне інформаційне повідомлення.

2. ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧІ

Front-end і back-end – це дві основні складові будь-якого веб-додатку. Вони виконують різні функції і взаємодіють між собою, щоб забезпечити повноцінне функціонування додатку.

2.1. Front-end

Front-end – це частина веб-додатку, з якою взаємодіє користувач. Вона відповідає за те, щоб представити дані та інтерфейс користувачу у вигляді веб-сторінок.

Він може бути реалізований за допомогою різних технологій, таких як HTML, CSS/SCSS і JavaScript/TypeScript. Популярними фреймворками та бібліотеками для розробки фронтенду є React, Angular та Vue.js.

Front-end відповідає за відображення контенту, стилізацію інтерфейсу користувача, обробку подій (наприклад, кліків миші або введення даних), валідацію форм, взаємодію з користувачем та відправку запитів на back-end для отримання або збереження даних.

2.1.1. React

React JS – це інтерфейсна бібліотека, яка стала популярним середовищем для сучасної веб-розробки у спільноті програмування JavaScript. Вона використовується для швидкого та ефективного створення інтерактивних користувацьких інтерфейсів і веб-додатків із застосуванням значно меншої кількості коду, ніж під час використання звичайного JavaScript. Замість того щоб розглядати весь призначений для користувача інтерфейс як єдине ціле, React.js пропонує розробникам розділити ці складні призначені для користувача інтерфейси на окремі компоненти, які можна використовувати повторно і які утворюють блоки всього призначеного для користувача інтерфейсу [6]. Таким чином, ця бібліотека надає можливість оновлювати інтерфейс динамічно без перезавантаження сторінки, що в свою чергу дозволяє створювати інтерактивні додатки, які реагують на дії користувача миттєво. Крім того, React використовує

віртуальний DOM для оптимізації процесу оновлення компонентів сторінки. Він зберігає копію реального DOM у пам'яті та оновлює його ефективно, що позитивно відображається на швидкодії веб-додатку.

2.1.1.1. React Router

React Router – це бібліотека для навігації між різними частинами (сторінками) React-додатку. Вона дозволяє встановлювати правила маршрутизації і відображати компоненти залежно від URL-адреси [7]. Саме React Router забезпечить можливість переходити між сторінками веб-сайта (навігацію) веб-додатку без перезавантаження сторінки.

2.1.1.2. React Content Loader

Для покращення користувацького досвіду, в момент поки з бек-енду завантажуються необхідні товари, замість порожньої сторінки покупець повинен бачити так звані «заглушки» для контенту. Саме React Content Loader забезпечить можливість розмістити їх на сторінці і демонструвати користувачеві.

2.1.1.3. React Paginate

Для покращення користувацького досвіду, всі товари інтернет-магазину не варто демонструвати на одній нескінченній сторінці. Необхідно забезпечити розбивку такої сторінки та надати користувачеві можливість вільно навігувати між ними. Цього можна досягти використовуючи бібліотеку React Paginate.

2.1.2. Redux

Redux – це бібліотека для управління станом додатка в JavaScript-додатках, особливо тих, що базуються на бібліотеці React. Вона дозволяє зберігати стан додатка в одному централізованому місці і керувати ним за допомогою спеціальних функцій. Стан всього додатка зберігається в єдиному об'єкті, який називається "store". Це дозволяє уникнути проблем з розділеним та непередбачуваним станом у складних додатках. Стан є незмінним. Це означає, що замість того, щоб змінювати його безпосередньо, завжди повертається новий об'єкт стану після кожної зміни. Це дозволяє легше відслідковувати зміни та

уникнути неочікуваних побічних ефектів [8]. Зміни стану додатка в Redux здійснюються за допомогою "дій" (actions) та "редукторів" (reducers). Дії – це прості об'єкти, які описують, що саме відбувається (наприклад, "додати товар до кошику"). Редуктори – це чисті функції, які обробляють ці дії та змінюють стан відповідно до них [9].

2.1.3. QS

QS – це бібліотека JavaScript, яка використовується для роботи з рядками запитів (query strings) у веб-додатках. Вона надає можливість розбирати рядки запитів з URL-адреси та формувати рядки запитів для включення в URL [10]. Саме ця бібліотека забезпечить зберігання параметрів пошуку, сортування, пагінації в адресі URL для відправлення запитів на бекенд і отримання відповідних результатів.

2.1.4. SASS

SASS (Syntactically Awesome Style Sheets) – це метамова для CSS, яка додає додаткові функції і можливості до стандартного CSS. Вона надає розширений синтаксис, який дозволяє використовувати змінні, вкладені правила, міксини, спадкування та інші корисні функції [11]. SASS дозволяє писати більш структурований, організований та підтримуваний CSS-код, що полегшує розробку та супровід проектів.

2.2. Back-end

Back-end – це серверна частина веб-додатка, яка відповідає за обробку запитів від фронтенду та виконання різних функцій, таких як обробка даних, доступ до бази даних, автентифікація користувачів тощо. Back-end може бути написаний за допомогою різних мов програмування та фреймворків, таких як Node.js з Express.js, Python з Django або Flask, PHP з Laravel тощо.

Back-end відповідає за обробку запитів, роботу з базою даних, бізнес-логіку додатка, забезпечення безпеки, збереження даних тощо.

2.2.1. MockAPI

MockAPI – це інструмент, який дозволяє легко створювати віртуальні API для тестування та прототипування програмного забезпечення. Інтерфейс MockAPI дуже простий та інтуїтивно зрозумілий. Він забезпечує можливість швидко створювати нові ендпоінти, визначати, які дані вони повертають та як вони обробляють запити [12]. Саме MockAPI забезпечить зберігання товарів веб-додатка без необхідності налаштування та роботи з реальним сервером.

2.2.2. Axios

Axios – це бібліотека для виконання HTTP-запитів у середовищі JavaScript, яка може працювати як у браузері, так і на сервері. Вона надає простий та зрозумілий API для взаємодії з HTTP-серверами та отримання або надсилання даних [13]. В комбінації з бібліотекою QS вона забезпечить можливість відправляти get-запити на MockAPI, враховуючи параметри пошуку та/або сортування та/або пагінації, які необхідні користувачеві веб-сайта.

2.2.3. Vercel

Vercel – це хмарний провайдер, який спеціалізується на хостингу веб-додатків та статичних сайтів [14]. Він дозволить розгорнути веб-додаток швидко, легко та безпечно. Vercel інтегрується з репозитарієм Git, за рахунок чого для розгортання додатку не потрібно буде робити жодних налаштувань. Також, у разі змін в репозитарії, Vercel автоматично їх синхронізує та пересобере веб-додаток.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1. Ініціалізація проекту (Create React App)

create-react-app – це інструмент командного рядка, який дозволяє швидко створити новий проект React з попередньо налаштованою конфігурацією. Він автоматично запобігає виникненню невідповідностей версій компонентів, що використовуються під час створення проекту React [15]. Для його використання, необхідно відкрити термінал у директорії, де планується створити додаток і виконати код, в якому викликати цей інструмент і вказати назву проекту, що створюється (рис. 3.1):

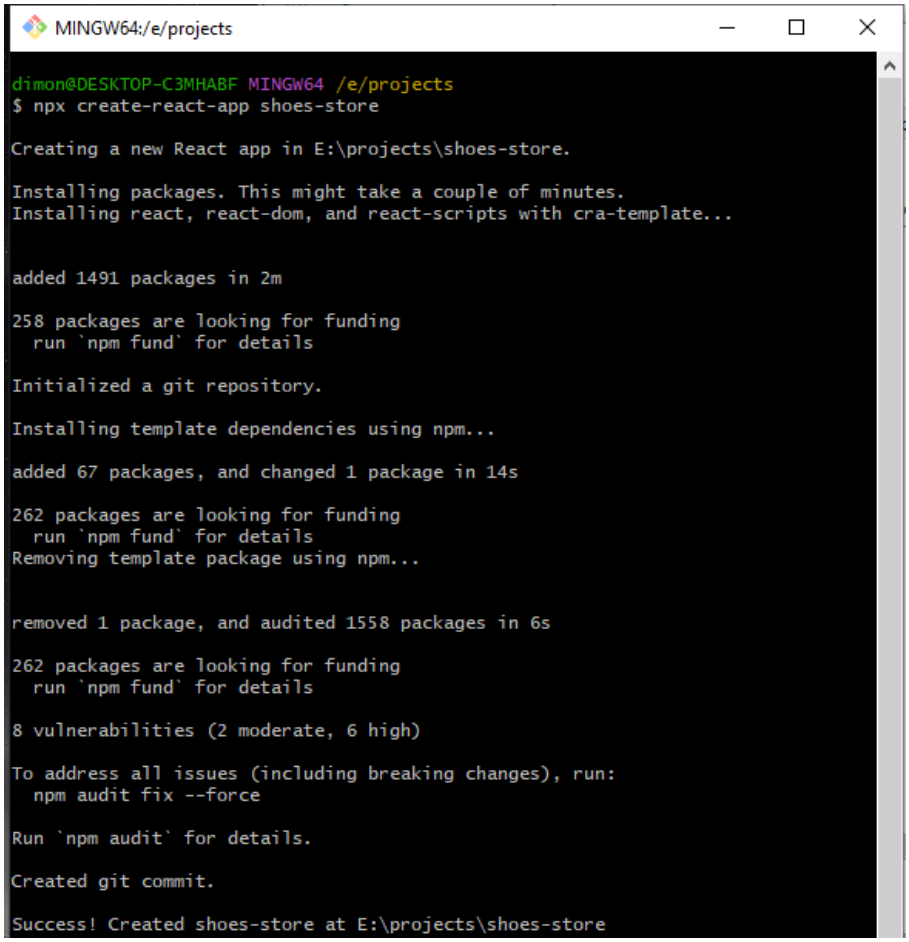


Рисунок 3.1 – Виконання команди для створення React проекту

Таким чином було створено директорію для зберігання всіх необхідних файлів та автоматично ініціалізовано систему контролю версій. Необхідно здійснити перевірку працездатності React-додатку, виконавши наступний код в терміналі Visual Studio Code (рис. 3.2):

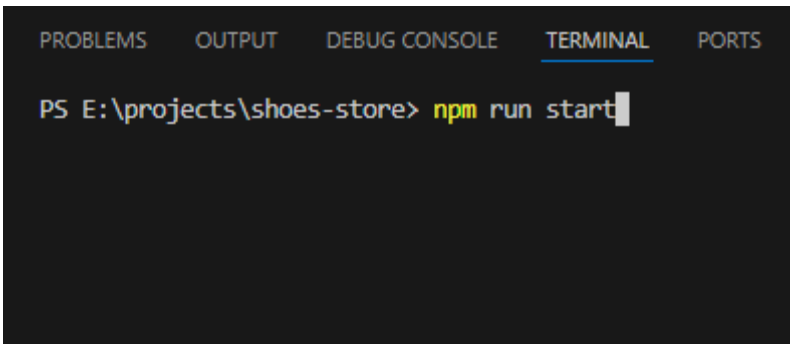


Рисунок 3.2 – Запуск React-додатку на локальному сервері

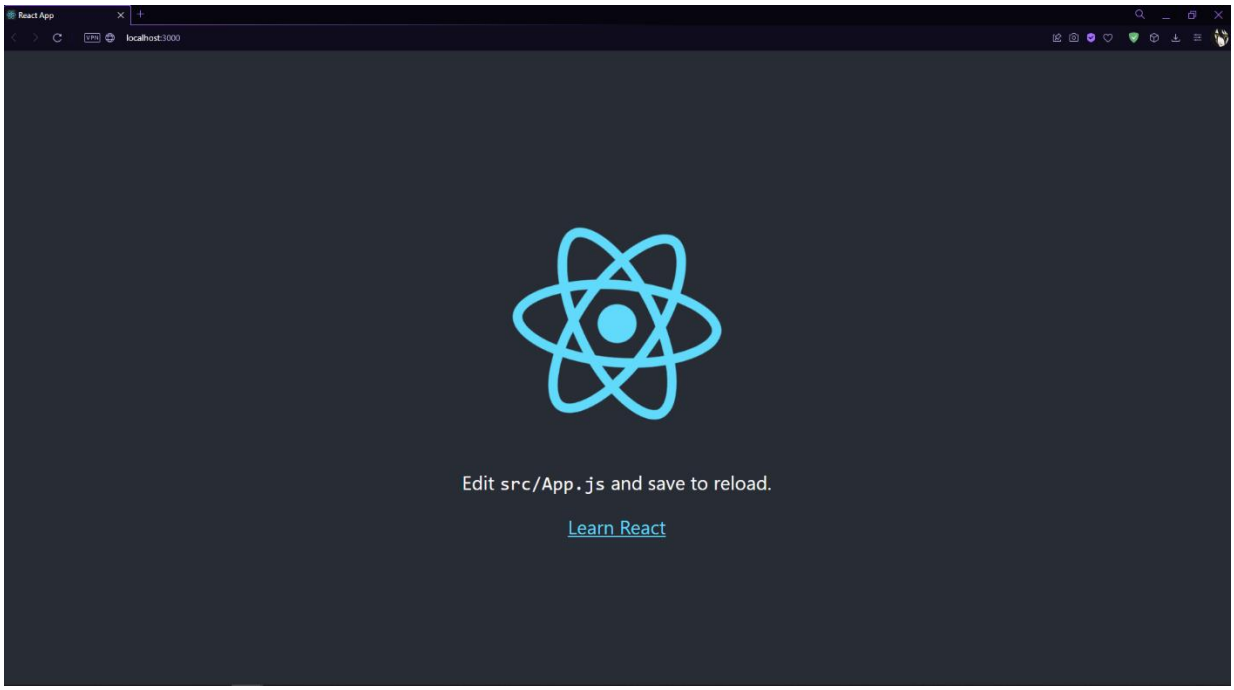


Рисунок 3.3 – Вигляд головної сторінки React-додатку

React-застосунок працює, про що свідчить сторінка, що відкрилася. Далі, можна розвивати веб-додаток.

3.2. Підготовка бекенду

3.2.1. Підготовка інформації про товари

Вся інформація про товари, яку буде отримувати веб-додаток, зберігатиметься у спеціальному .json файлі. Для збереження інформації про продукцію знадобляться наступні поля (рис. 3.4):

Рисунок 3.4 – Поля .json файлу, що використовується для зберігання інформації про товари

Опис використовуваних полів наведено в таблиці 3.1.

Таблиця 3.1 Опис полів, що використовуються для збереження інформації про товари

Назва поля	Тип даних	Опис поля
id	рядок	Унікальний ідентифікаційний номер товару. Примітка: для MockAPI важливо щоб id був саме рядком, а не числом, інакше він не зможе повернути конкретні товари, які запитуються по id
imageUrl	рядок	Посилання на зображення продукції
title	рядок	Назва продукції
sizes	масив чисел	Список доступних розмірів продукції (взуття)
price	число	Ціна продукції
category	число	Номер категорії. Примітка: номери відповідають наступним категоріям: 1 – Кросівки 2 – Кеди 3 – Черевики 4 – Шльопанці

Продовження таблиці 3.1

rating	число	Популярність товару
description	рядок	Опис товару

Для демонстрації можливостей веб-додатка було сформовано .json файл, що складається з 17-ти товарів.

Повний код файлу shoes.json наведено в **додатку А**.

3.2.2. Налаштування MockAPI

Авторизувавшись на сайті mockapi.io і створивши проект переходимо до його налаштувань (рис. 3.5):

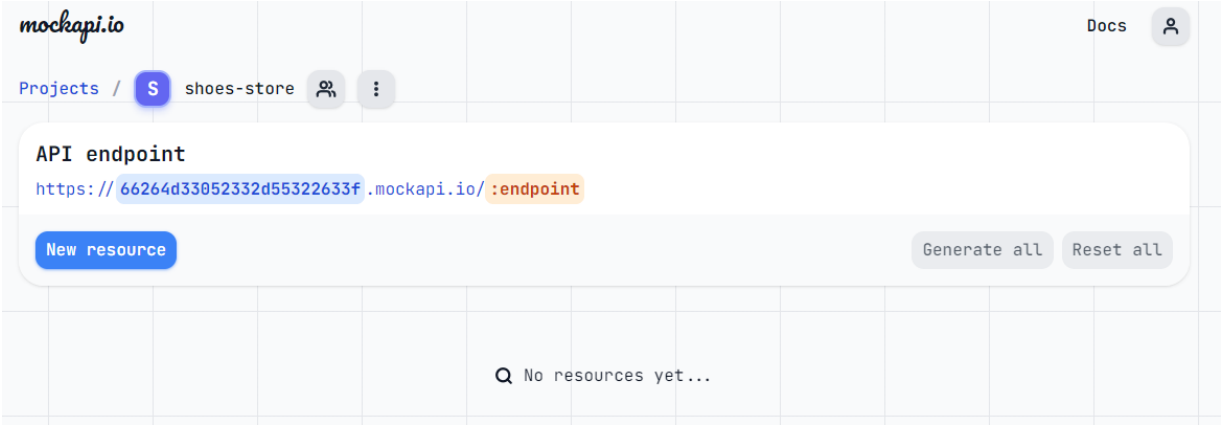


Рисунок 3.5 – Вигляд сторінки налаштування проекту MockAPI

Як можна побачити, MockAPI відразу створив endpoint для проекту, і саме за цим посиланням будуть відправлятися get-запити для отримання інформації про товари. Але для початку необхідно забезпечити зберігання цієї інформації в межах MockAPI. Для цього на сторінці проекта необхідно натиснути кнопку New resource і призначити йому ім’я «items», після чого натиснути кнопку Create (рис. 3.6):

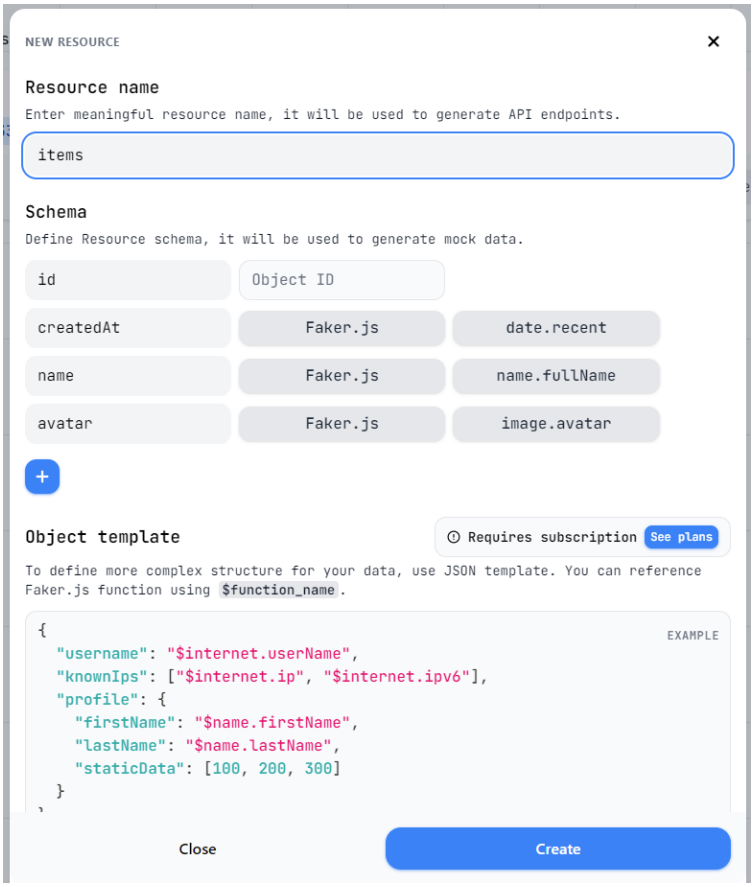


Рисунок 3.6– Вікно для створення сховища

Як можна побачити, зовнішній вигляд головної сторінки оновився (рис. 3.7) і тепер стає доступна взаємодія зі сховищем.

Рисунок 3.7 – Вигляд сторінки налаштування проекту MockAPI після створення сховища

Щоб зберегти інформацію про товари у сховищі MockAPI, необхідно натиснути кнопку Data, і в поле вікна, що з’явилося скопіювати сформований раніше .json файл, після чого натиснути кнопку Update (рис. 3.8):

Рисунок 3.8 – Збереження інформації про товари у сховищі

Таким чином, було підготовлено MockAPI для подальшого надсилання get-запитів і відповідно отримання товарів у веб-додаток, що створюється.

3.3. Створення глобальних станів і налаштування їх сховища

Для створення станів і управління ними спочатку необхідно додати до проекту бібліотеки, що забезпечать таку можливість, а саме @reduxjs/toolkit (що забезпечить функціонал станів), react-redux (що забезпечить інтеграцію бібліотеки redux у react додаток) та axios (що буде використана для відсилання get-запитів на бекенд). Додати їх можна виконавши відповідний код у терміналі Visual Studio Code (рис. 3.9 - 3.10):

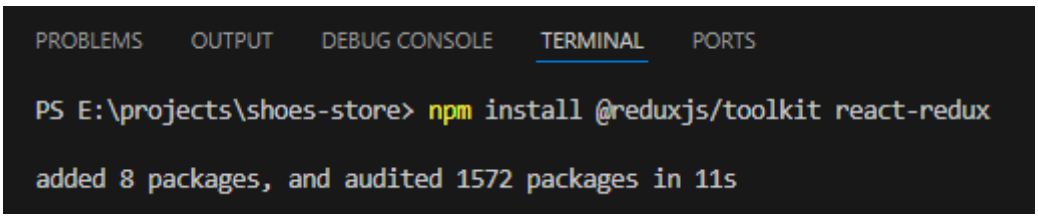


Рисунок 3.9 – Встановлення бібліотек redux та react-redux

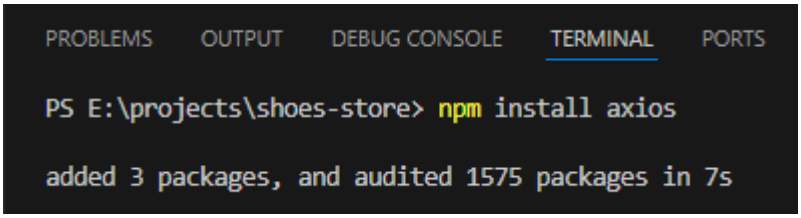


Рисунок 3.10 – Встановлення бібліотеки axios

Таким чином, бібліотеки додано до проекту і вони готові до використання.

3.3.1. Стан товарів

Для управління станом товарів, для початку ці товари необхідно отримати з бекенду у вигляді об’єктів. Для цього використаємо бібліотеку Axios (рис. 3.11):

```
export const fetchShoes = createAsyncThunk('shoes/fetchShoesStatus', async (params, thunkAPI) => {
  const { sortBy, order, category, search, currentPage } = params;
  const { data } = await axios.get(
    `https://66264d33052332d55322633f.mockapi.io/items?page=${currentPage}&limit=8&${category}&sortBy=${sortBy}&order=${order}&${search}`,
  );
  return data;
});
```

Рисунок 3.11 – Код дії fetchShoes

Було визначено асинхронну дію fetchShoes, що створена за допомогою createAsyncThunk, в якій виконується GET-запит до віддаленого сервера, і, як результат, ця дія повертає отримані з бекенду об’єкти.

Далі, необхідно створити так званий “slice” – функція, яка, як слідує з назви, вирізає частину даних додатка, і визначає як ці дані будуть оновлюватися у відповідь на дії (actions) (див.рисунок 3.12):

```
const initialState = {
  items: [],
  status: 'loading',
};

const shoesSlice = createSlice({
  name: 'shoes',
  initialState,
  reducers: {
    setItems(state, action) {
      state.items = action.payload;
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(fetchShoes.pending, (state) => {
        state.status = 'loading';
        state.items = [];
      })
      .addCase(fetchShoes.fulfilled, (state, action) => {
        state.items = action.payload;
        state.status = 'success';
      })
      .addCase(fetchShoes.rejected, (state) => {
        state.status = 'error';
        state.items = [];
      });
  },
});
```

Рисунок 3.12 – Код слайсу для управління товарами

Таким чином було створено slice під назвою “shoesSlice”.

В ньому визначено:

- Назву слайса – “shoes”
- Початковий стан – ініціалізується за межами виклику функції createSlice для покращення читання коду.
- Набір редукторів (reducers), що включає в себе setItems, який використовується для оновлення масиву товарів.
- Обробники додаткових дій (extraReducers), що включають обробку станів pending, fulfilled і rejected для асинхронної дії fetchShoes. Це зроблено для відслідковування статусу отримання товарів з бекенду, і, таким чином, поки

вони отримуються або вони не були отримані (з будь-яких причин) – замість простого undefined станом товарів буде пустий масив, що в свою чергу не порушуватиме роботу візуальної частини додатка (оскільки відображення товарів на сторінках і функції пов'язані з ними очікуватимуть в якості аргументів саме масиви об'єктів, навіть якщо вони порожні).

Повний код файлу shoesSlice.js наведено в **додатку Б**.

3.3.2. Стан кошика

Для контролю стану кошика, аналогічно до стану товарів, необхідно створити відповідний slice (рис. 3.13):

```
const cartSlice = createSlice({
  name: 'cart',
  initialState,
  reducers: {
    addItem(state, action) {
      const findItem = state.items.find((obj) => {
        return obj.id === action.payload.id && obj.size === action.payload.size;
      });
      if (findItem) {
        findItem.count++;
      } else {
        state.items.push({
          ...action.payload,
          count: 1,
        });
      }
      state.totalPrice = calcTotalPrice(state.items);
    },

    removeItem(state, action) {
      state.items = state.items.filter(
        (obj) => !(obj.id === action.payload.id && obj.size === action.payload.size),
      );
      state.totalPrice = calcTotalPrice(state.items);
    },

    clearItems(state) {
      state.items = [];
      state.totalPrice = 0;
    },

    minusItem(state, action) {
      const findItem = state.items.find((obj) => {
        return obj.id === action.payload.id && obj.size === action.payload.size;
      });
      console.log(findItem);
      if (findItem && findItem.count > 1) {
        findItem.count--;
        state.totalPrice = state.totalPrice - findItem.price;
```

Рисунок 3.13 – Код слайсу для управління кошиком товарів

В слайсі cartSlice визначено:

- Назву слайса – “cart”
- Початковий стан – ініціалізується за межами виклику функції createSlice для покращення читання коду, містить в собі об’єкти, що знаходяться у кошику та їх загальну вартість.
- Набір редукторів (reducers), який включає в себе addItem (для додавання товару в кошик або збільшення його кількості в кошику), removeItem (для видалення товару з кошика), clearItems (для очистки кошика від всіх товарів) та minusItem (для зменшення кількості товарів в кошику)

Повний код файлу cartSlice.js наведено в **додатку В**.

3.3.3. Стан фільтрації

Для контролю стану фільтрації (визначення параметрів фільтрації), аналогічно до стану товарів і стану кошика, необхідно створити відповідний slice (рис. 3.14):

```
const initialState = {
  searchValue: '',
  categoryId: 0,
  currentPage: 1,
  sort: {
    name: 'за популярністю',
    sortProperty: 'rating',
  },
};

const filterSlice = createSlice({
  name: 'filters',
  initialState,
  reducers: {
    setCategoryId(state, action) {
      state.categoryId = action.payload;
    },
    setSearchValue(state, action) {
      state.searchValue = action.payload;
    },
    setSort(state, action) {
      state.sort = action.payload;
    },
    setCurrentPage(state, action) {
      state.currentPage = action.payload;
    },
  },
});
```

Рисунок 3.14 – Код слайсу для управління параметрами фільтрації

В слайсі filterSlice визначено:

- Назву слайса – “cart”.
- Початковий стан – ініціалізується за межами виклику функції createSlice для покращення читання коду, містить в собі значення, введене в поле пошуку, номер категорії (0 – показати всі товари у всіх категоріях), номер сторінки пагінації, параметри сортування.
- Набір редукторів (reducers), який включає в себе setCategoryId (для оновлення номеру категорії), setSearchValue (для оновлення інформації щодо вмісту поля пошуку), setSort (для оновлення параметрів сортування) та setCurrentPage (для оновлення номеру сторінки пагінації).

Повний код файлу filterSlice.js наведено в **додатку Г**.

3.3.4. Налаштування сховища

Для використання створених слайсів бібліотека Redux вимагає налаштувати і створити сховище (Redux Store) [16]. Таке налаштування і створення виконується однією функцією configureStore() (рис. 3.15):

```
1  import { configureStore } from '@reduxjs/toolkit';
2  import filter from './slices/filterSlice';
3  import cart from './slices/cartSlice';
4  import shoes from './slices/shoesSlice';
5
6  export const store = configureStore({
7    reducer: {
8      filter,
9      cart,
10     shoes,
11   },
12 });
```

Рисунок 3.15 – Код для налаштування сховища слайсів

Аргументами в цю функцію ми передаємо редусери раніше створених слайсів товарів, кошика, фільтрації. Для того, щоб веб-додаток міг використовувати це сховище, у файлі index.js, який відповідає за рендер всього

react-додатку, у функції рендеру необхідно надати кореневому компоненту доступ до сховищу, використовуючи функцію бібліотеки react-redux під назвою Provider (рис. 3.16):

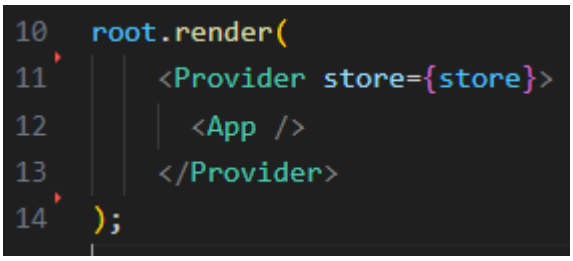


Рисунок 3.16 – Код для забезпечення зв'язку додатка зі сховищем слайсів

Таким чином, Provider, як випливає з назви, забезпечує зв’язок всього додадтка зі сховищем слайсів.

Повний код файлу store.js наведено в **додатку Г**.

Повний код файлу index.js наведено в **додатку 6**.

3.4. Створення компонентів та розміщення їх на сторінках

3.4.1. Компонент Header

Веб-сайт, що розробляється, в своєму хедері має містити логотип і назву інтернет-магазину в лівій частині, кнопку переходу до кошика, що розміщена в правій частині та яка має показувати скільки товарів додано до кошика та їх загальну вартість, при чому ця кнопка повинна бути прихована, коли користувач знаходиться на сторінці кошика, і поле пошуку, що розміщене в центральній частині хедера, при чому на сторінках кошика, товару та оформлення замовлення таке поле пошуку має бути прихованим. Розглянемо розмітку цього компонента (рис. 3.17):

```
return (
  <div className="header">
    <div className="container">
      <Link to="/">
        <div className="header__logo">
          <img width="38" src={logoSvg} alt="Shoes Store logo" />
          <div>
            <h1>Shoes Store</h1>
            <p>Крокуй стильно!</p>
          </div>
        </div>
      </Link>
      {!containsSubstring('/cart') &&
        !containsSubstring('/shoes') &&
        !containsSubstring('/checkout') && <Search />}
      <div className="header__cart">
        {location.pathname !== '/cart' && (
          <Link to="/cart" className="button button--cart">
            <span>{totalPrice} ₴</span>
            <div className="button__delimiter"></div>
            <svg ···
            </svg>
            <span>{totalCount}</span>
          </Link>
        )}
      </div>
    </div>
  </div>
);
```

Рисунок 3.17 – Код розмітки компонента header

Розглянемо ключові частини:

- – це контейнер для всього вмісту хедера.
- <Link to="/"> – компонент для створення посилання на головну сторінку сайту. Використовується з бібліотеки React Router, яка забезпечує навігацію додатка. Таким чином, користувач, натиснувши на логотип або назву інтернет-магазину перейде на його головну сторінку.
- – контейнер для логотипу та назви інтернет-магазину і його слогану.
- - зображення логотипу, logoSvg - це змінна, що містить шлях до SVG-зображення.
- !containsSubstring('/cart') && !containsSubstring('/shoes') && !containsSubstring('/checkout') && <Search /> – це умова, що дозволить приховати поле пошуку на сторінках кошика, товару та оформлення замовлення.

- – контейнер для блоку кошика.
- {location.pathname !== '/cart' && (…) – це умова, що дозволить приховати кнопку переходу до кошика, коли користувач знаходиться на сторінці кошика.
- <Link to="/cart" className="button button--cart"> – компонент для створення посилання на сторінку кошика, стилізований під кнопку.
- {totalPrice} ₴ – відображення загальної суми покупок у кошику в гривнях, обчислюється в окремій функції і зберігається у змінній totalPrice
- – роздільна лінія між сумою та іконкою кошика.
- <svg>...</svg> – іконка кошика у вигляді SVG-зображення.
- {totalCount} – відображення загальної кількості товарів у кошику, обчислюється в окремій функції і зберігається у змінній totalCount.

Рисунок 3.18 – Вигляд хедеру за звичайних умов

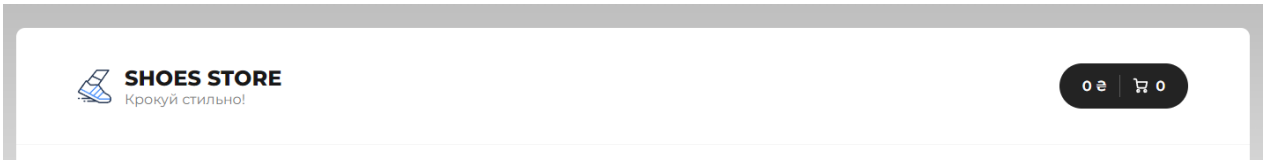


Рисунок 3.19 – Вигляд хедеру при умові прихованого компонента пошуку

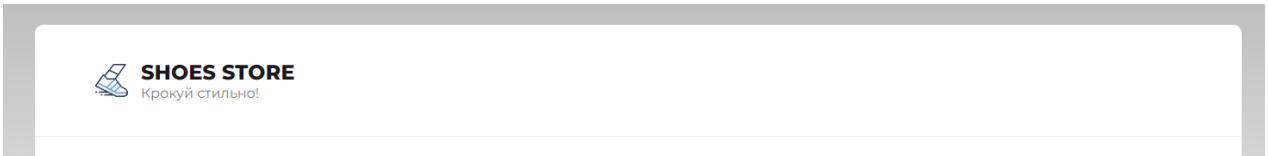


Рисунок 3.20 – Вигляд хедеру при умові прихованих компоненту пошуку та кнопки переходу до кошику

Таким чином, було реалізовано хедер, що містить необхідні компоненти, які приховуються за потрібних умов.

Повний код файлу Header.jsx наведено в **додатку Д**.

3.4.1.1. Компонент пошуку

Поле пошуку має забезпечити користувачеві можливість пошуку товарів за їх назвою. Для зручності користувача, необхідно також реалізувати кнопку для очищення поля пошуку. Отже, розглянемо розмітку цього компоненту (рис. 3.21):

```
40  return (
41    <div className={styles.root}>
42      <img className={styles.icon} src={searchIcon} alt="Пошук"></img>
43      <input
44        ref={inputRef}
45        value={value}
46        onChange={onChangeInput}
47        className={styles.input}
48        placeholder="Знайти товар..."
49      />
50      {value && (
51        <img
52          onMouseDown={onMouseDownClear}
53          onClick={onClickClear}
54          className={styles.clearIcon}
55          src={clearIcon}
56          alt="Очистити поле пошуку"></img>
57      )}
58    </div>
59  );
```

Рисунок 3.21 – Код розмітки компонента пошуку

Розглянемо ключові частини:

- – це контейнер для всього поля пошуку, де styles.root вказує на клас CSS для стилізації цього контейнера.
- <img className={styles.icon} src={searchIcon} alt="Пошук"></img> – зображення іконки пошуку, яке відображається перед полем вводу. Клас styles.icon вказує на стилізацію цієї іконки через CSS. searchIcon - це шлях до зображення іконки пошуку.
- <input> - це саме поле вводу для пошуку. Він має наступні атрибути:
  - ref={inputRef} – цей атрибут використовується для посилання на DOM-елемент (поле вводу) у коді React. В данному випадку це необхідно для стилізації, коли користувач натискає на поле пошуку (фокусується на ньому) кордон цього поля стає більш темним.
  - value={value} – вміст поля вводу, який зберігається у стані компонента (використовується React.Usestate. Він дозволяє компонентам React

зберігати стан і оновлювати його [17]. Можна сказати, що це глобальна змінна, яку можна зручно зберігати, оновлювати та використовувати у функціях).

- onChange={onChangeInput} – ця функція викликається при зміні значення в полі вводу. Вона оновлює стан змінної value, яка була описана раніше, зберігаючи в ній поточний вміст поля <input>, та викликає функцію updateSearchValue (рис. 3.22):

```
29  const onChangeInput = (event) => {
30    setValue(event.target.value);
31    updateSearchValue(event.target.value);
32  };
33
34  const updateSearchValue = React.useCallback(
35    debounce((str) => {
36      dispatch(setSearchValue(str));
37    }, 300),
38    [],
39  );
```

Рисунок 3.22 – Код функцій onChangeInput та updateSearchValue

Функція updateSearchValue, в свою чергу викликає оновлення стану фільтрації, слайс якого було створено і описано раніше (3.3.3 Стан фільтрації), що в свою чергу спричинить зміну get-запиту на бекенд, з метою повернути з нього саме той товар, який шукає користувач. Крім того, використовується функція з бібліотеки Lodash, а саме debounce. Це дозволяє забезпечити те, що запити на бекенд не будуть відправлятися при кожній зміні в полі пошуку, а лише через 300 мс після того, як користувач перестане вводити текст. Такий підхід допомагає уникнути зайвих запитів до бекенду, які в свою чергу можуть призвести до тимчасового блокування додатка з боку MockAPI і відповідно недоступності товарів.

- className={styles.input} – цей атрибут вказує на клас CSS для стилізації поля вводу.
- placeholder="Знайти товар..." - це підказка, яка відображається в полі вводу, коли воно порожнє.

- {value && (...)} – це умовний блок, який дозволить, використовуючи стан поля пошуку value приховати кнопку для очистки поля пошуку, якщо воно порожнє.
- <img> – зображення іконки очищення поля вводу. Клас styles.clearIcon вказує на стилізацію цієї іконки через CSS. clearIcon – це змінна, що зберігає шлях до зображення іконки очищення поля вводу.
- onClick={onClickClear} – ця функція викликається при кліканні на іконці очищення поля вводу. Змінює стан значення поля пошуку (змінна value) на ‘’ (порожній рядок) і оновлює стан слайсу фільтрації, а саме поля searchValue, що забезпечить відображення всіх товарів.

Рисунок 3.23 – Вигляд порожнього поля пошуку

Рисунок 3.24 – Вигляд поля пошуку при введеному в ньому тексті

Таким чином, реалізоване поле пошуку, яке при взаємодії дещо змінює свій стиль і показує кнопку для очистки поля.

Повний код файу /Search/index.jsx наведено в **додатку Є**.

3.4.2. Компонент Footer

Футер, як правило використовується для розміщення додаткової інформації веб-додатка. Розглянемо розмітку цього компоненту (рис. 3.25):

```jsx
import React from 'react';

function Footer() {
  return (
    <div className="footer">
      <div className="container">
        <p>Роботу виконав Білецький Дмитро. Всі права захищенні. СумДУ 2024</p>
      </div>
    </div>
  );
}

export default Footer;

```

Рисунок 3.25 – Код розмітки компоненту footer

В собі він має єдину інформацію про автора роботи.

Рисунок 3.26 – Вигляд компоненту footer

Повний код файлу Footer.jsx наведено в **додатку Ж**.

3.4.3. Домашня сторінка

Головна сторінка веб-сайту, що реалізується має містити каталог товарів, категорії товарів з можливістю змінювати їх, випадаючий блок з параметрами сортування товарів, та має бути реалізовано пагінацію, тобто розбивку головної сторінки з обмеженням кількості товару, що показується. Розглянемо розмітку такої сторінки (рис. 3.27):

```jsx
return (
  <div className="container">
    <div className="content__top">
      <Categories value={categoryId} onChangeCategory={onChangeCategory} />
      <Sort setOrder={setOrder} />
    </div>
    <h2 className="content__title">Каталог</h2>
    {status === 'error' && (
      <div className="content__error-info">
        <h2>Сталася помилка 😕</h2>
        <p>
          На жаль, не вдалося знайти моделі взуття. Перевірте правильність пошукового запиту або
          зверніться в службу підтримки.
        </p>
      </div>
    )}
    <div className="content__items">{status === 'loading' ? skeletons : shoes}</div>
    <Pagination currentPage={currentPage} onChangePage={onChangePage} />
  </div>
);
```

Рисунок 3.27 – Код розмітки домашньої сторінки

Розглянемо ключові частини:

- – В цьому блоці розміщуються такі компоненти як категорії та сортування:
  - <Categories value={categoryId} onChangeCategory={onChangeCategory} /> – Цей компонент відображає список категорій товарів і передає поточне значення категорії (categoryId) і функцію для зміни категорії (onChangeCategory) (рис. 3.28):

```jsx
const onChangeCategory = React.useCallback((id) => {
  dispatch(setCategoryId(id));
  dispatch(setCurrentPage(1));
}, []);
```

Рисунок 3.28 – Код функції onChangeCategory

Функція onChangeCategory змінює стан слайсу фільтрації (3.3.3 Стан фільтрації), а саме значення поля categoryId, тим самим змінюючі параметри get-запиту, з метою відібрати лише ті товари, категорія яких цікавить користувача, та перенаправляє користувача на першу сторінку пагінації на випадок якщо він вирішив змінити категорію будучи на іншій по номеру сторінці.

- <Sort setOrder={setOrder} /> – Цей компонент дозволяє користувачеві вибрати порядок сортування товарів (за спаданням або за зростанням) і

передає функцію для встановлення порядку сортування (setOrder), яка являється функцією зміну стану (порядок сортування зберігається у вигляді стану (React.Usestate) для того щоб мати можливість змінювати цей стан через інші компоненти). Цю функцію необхідно передати в компонент <Sort />, оскільки саме там реалізована зміна порядку сортування товарів.

- {status === 'error' && (...)} – Це умовний оператор, який перевіряє, чи статус завантаження даних є помилковим. Якщо так, він відображає повідомлення про помилку.
- {status === 'loading' ? skeletons : shoes} – Це місце для відображення товарів або скелетонів, в залежності від статусу завантаження даних. Якщо дані все ще завантажуються (status === 'loading'), відображаються скелетони (заглушки) товарів, в іншому ж випадку буде відображено самі товари.
- <Pagination currentPage={currentPage} onChangePage={onChangePage} /> – Цей компонент відображає пагінацію і передає поточну сторінку (currentPage) і функцію для зміни сторінки (onChangePage) (рис. 3.29):

```
const onChangePage = (number) => {
  dispatch(setCurrentPage(number));
};
```

Рисунок 3.29 – Код функції onChangePage

Функція onChangePage змінює стан слайсу фільтрації, а саме поля currentPage, що спричинить зміну get-запиту, і покаже користувачеві товари, які знаходятся на сторінці, яку він обрав.

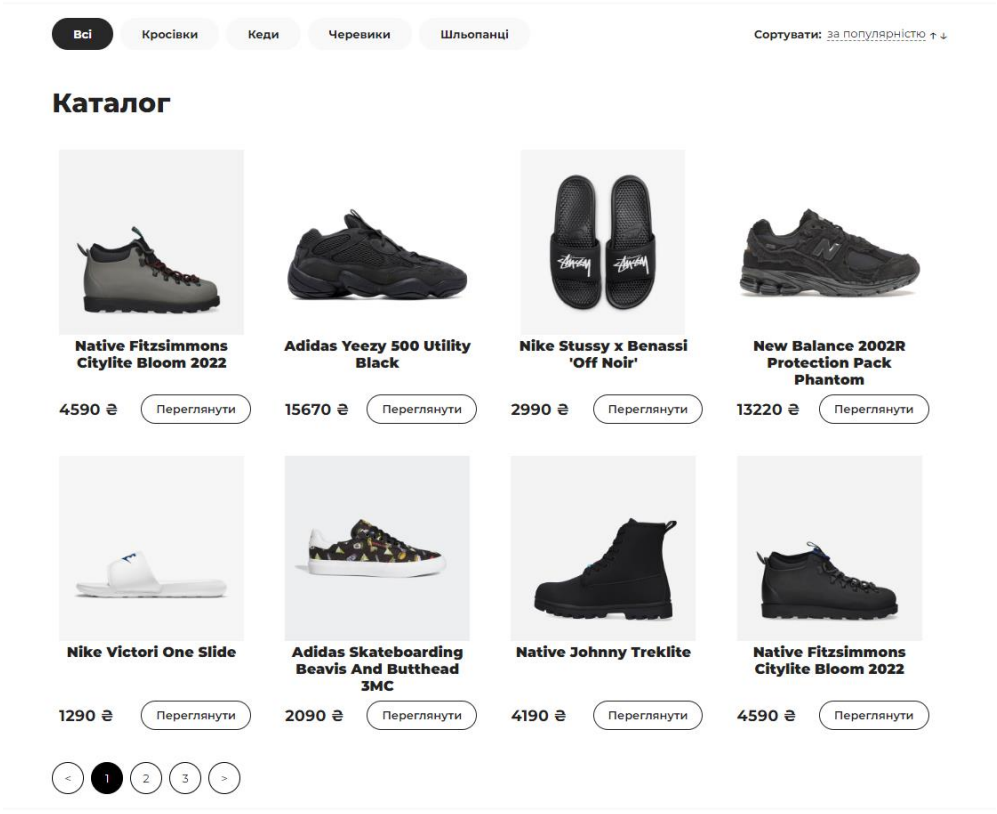


Рисунок 3.30 – Вигляд домашньої сторінки

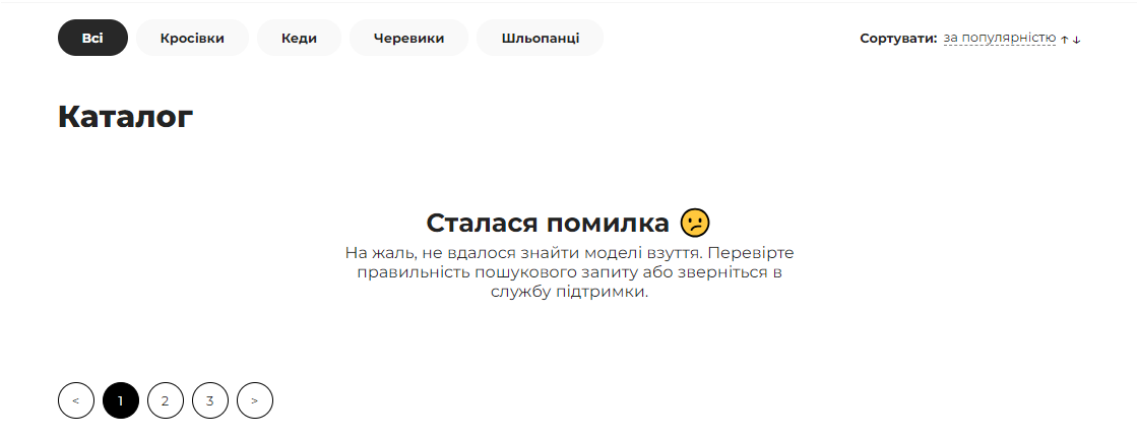


Рисунок 3.31 – Вигляд домашньої сторінки з повідомленням про помилку

Таким чином, реалізовано головну сторінку додатку, що містить в собі компоненти категорій, сортування і товарів, а у випадку, коли користувач ввів не правильний пошуковий запит або з інших причин з бекенду неможливо отримати товари – буде відображено інформаційне повідомлення.

Повний код файлу Home.jsx наведено в **додатку З**.

3.4.3.1. Компонент товарів

Функція, що викликає рендер товарів на домашній сторінці має наступний вигляд (рис. 3.32):

```
const shoes = items.map((obj) => <ShoesBlock {...obj} />);
```

Рисунок 3.**Error! No text of specified style in document.**32 – Код функції для відображення товарів на домашній сторінці

Для масиву items, в якому зберігаються всі товари, що надійшли з бекенду, викликається функція map(), яка для кожного елементу масиву (товару) викличе рендер компонента цього елемента (товара). Розглянемо розмітку компонента <ShoesBlock /> (рис. 3**Error! No text of specified style in document.**.33):

```
function ShoesBlock({ id, title, price, imageUrl }) {
  return (
    <div className="shoes-block-wrapper">
      <div className="shoes-block">
        <Link key={id} to={`/shoes/${id}`}>
          <img className="shoes-block__image" src={imageUrl} alt="Shoes" />
          <h4 className="shoes-block__title">{title}</h4>
          <div className="shoes-block__bottom">
            <div className="shoes-block__price">{price} ₴</div>
            <button className="button button--outline button--add">
              <span>Переглянути</span>
            </button>
          </div>
        </Link>
      </div>
    </div>
  );
}
```

Рисунок 3**Error! No text of specified style in document.**.33 – Код розмітки компоненту ShoesBlock

Розглянемо ключові частини:

- Функція ShoesBlock приймає об'єкт з властивостями id, title (назва товару), price (ціна товару та imageUrl (посилання на зображення товару) – саме ця інформація необхідна для демонстрації продукту користувачеві.

- Вміст блоку товару розміщується всередині , стиль якого допомагає розмістити блоки з товарами в межах сторінки.
- Елемент <Link> використовується для створення посилання на окрему сторінку товару, таким чином користувачеві не обов'язково натискати саме на кнопку Переглянути для переходу на сторінку продукту – достатньо натиснути в межах блоку товару, що цікавить. key={id} використовується для унікальності кожного елемента у списку товарів, щоб React міг ефективно відслідковувати їх зміни.
- <div className="shoes-block”> – містить всю інформацію про товар, а саме його зображення, назву, ціну.

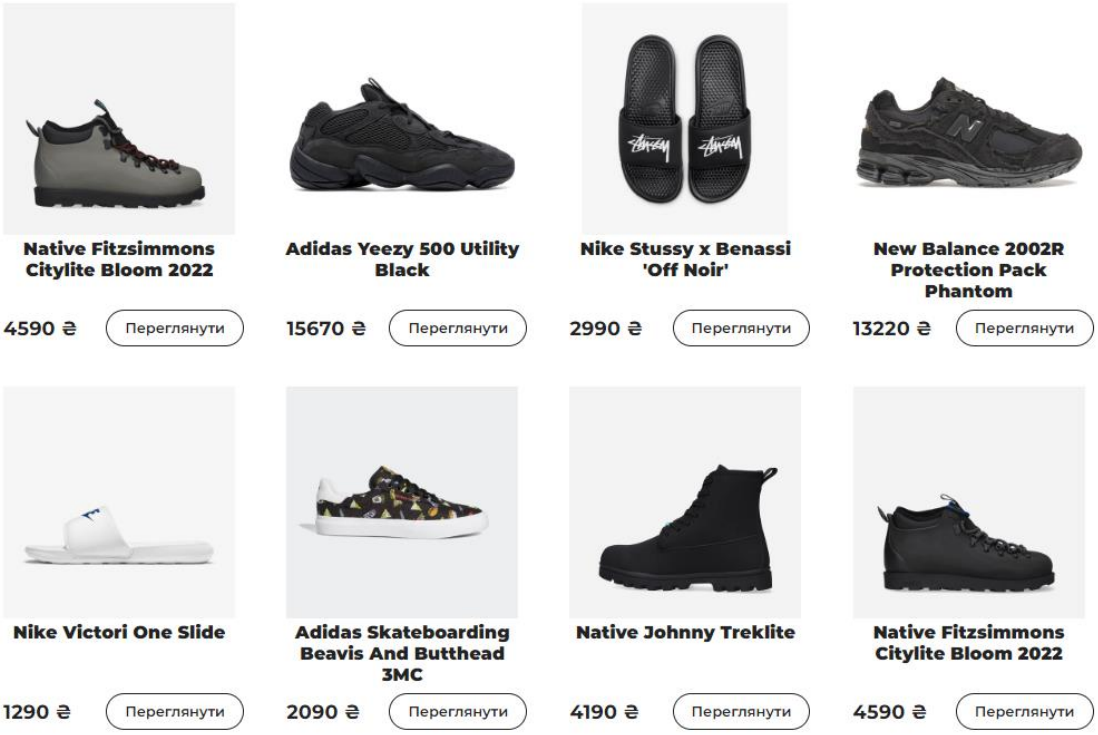


Рисунок 3**Error! No text of specified style in document.**.34 – Вигляд блоків для відображення товарів

Таким чином, було реалізовано відображення товарів, що отримуються з бекенду.

Повний код файлу /ShoesBlock/index.jsx наведено в **додатку И**.

3.4.3.2. Компонент скелетонів

Для забезпечення кращого користувацького досвіду в момент, поки дані все ще завантажуються, необхідно відображати скелетони (заглушки) товарів. Для цього необхідно використати бібліотеку React Content Loader. Для початку, її необхідно встановити у проект виконавши відповідний код в терміналі Visual Studio Code (рис. 3**Error! No text of specified style in document.**.35):

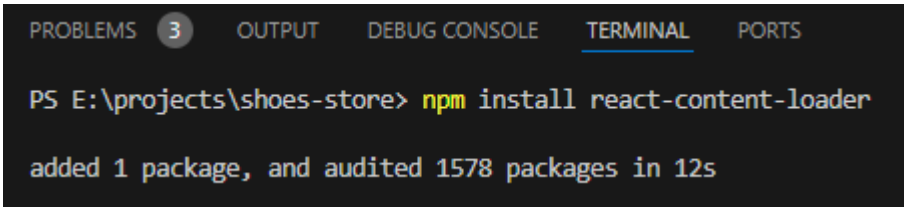


Рисунок 3**Error! No text of specified style in document.**.35 – Встановлення бібліотеки React Content Loader

Бібліотека встановлена і готова для використання. Для розмітки скелетонів було використано онлайн-конструктор https://skeletonreact.com [18] (рис. 3.36):

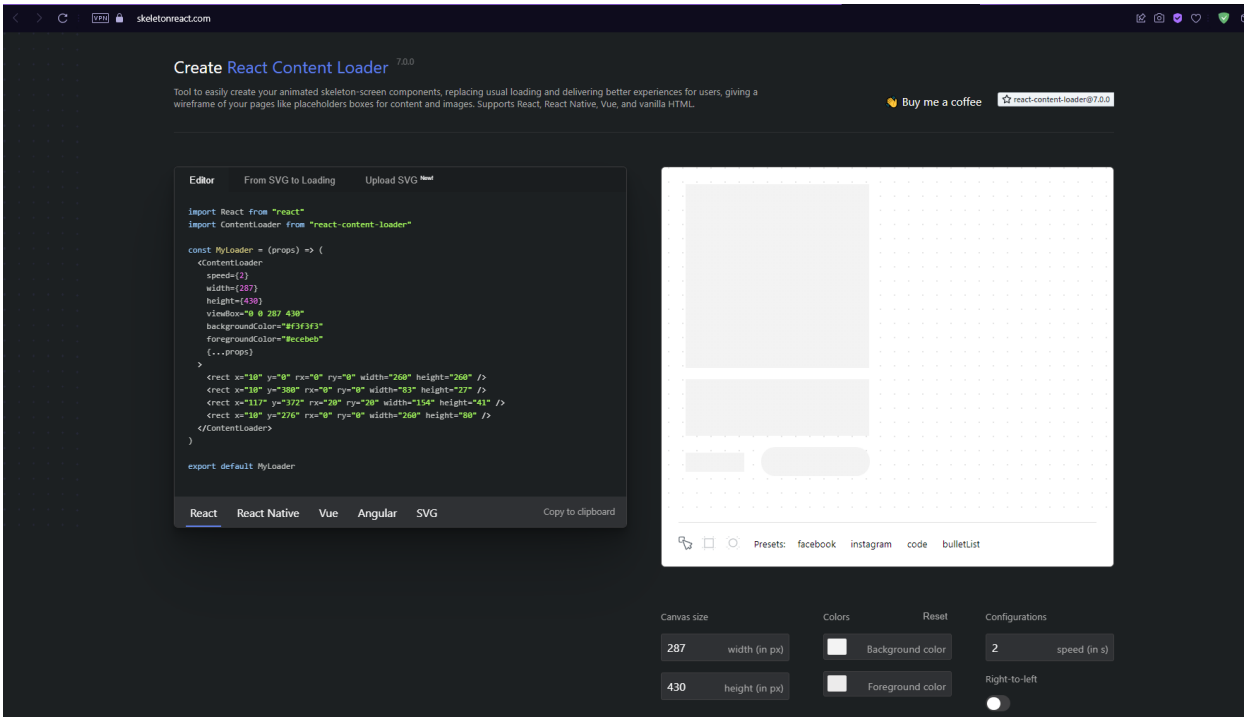


Рисунок **Error! No text of specified style in document.**3.36 – Вигляд онлайн-конструктора для створення скелетонів

Розмістивши компоненти скелетону, так, як того потребує веб-сайт, що розробляється, в лівій частині онлайн-конструктора було сформовано код, який було використано разом з бібліотекою React Content Loader (рис. 3**Error! No text of specified style in document.**.37):

```jsx
import React from 'react';
import ContentLoader from 'react-content-loader';

const Skeleton = (props) => (
  <ContentLoader
    speed={2}
    width={287}
    height={430}
    viewBox="0 0 287 430"
    backgroundColor="#f3f3f3"
    foregroundColor="#ecebeb"
    {...props}>
    <rect x="10" y="0" rx="0" ry="0" width="260" height="260" />
    <rect x="10" y="380" rx="0" ry="0" width="83" height="27" />
    <rect x="117" y="372" rx="20" ry="20" width="154" height="41" />
    <rect x="10" y="276" rx="0" ry="0" width="260" height="80" />
  </ContentLoader>
);

export default Skeleton;
```

Рисунок 3**Error! No text of specified style in document.**.37 – Код компонету скелетонів

Таким чином, сформовано компонент <Skeleton />, який буде рендеритися на місці кожного компоненту відображення товару, поки самі товари завантажуються з бекенду (рис. 3.38):

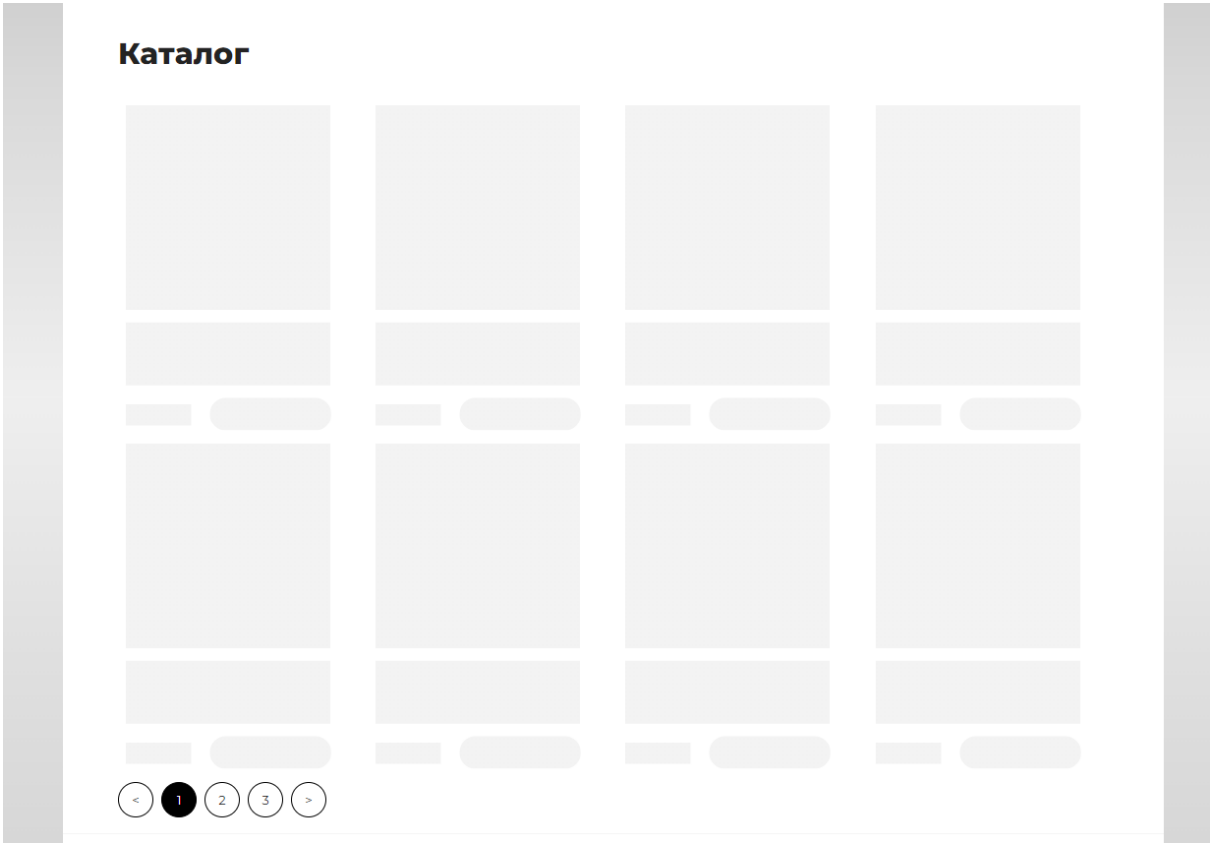


Рисунок **Error! No text of specified style in document.**3.38 – Вигляд скелетонів товарів

Повний код файлу /ShoesBlock/Skeleton.jsx наведено в **додатку І**.

3.4.3.3. Пагінація

Для реалізації функціоналу пагінації використаємо бібліотеку React Paginate. Для початку, її необхідно встановити у проект виконавши відповідний код в терміналі Visual Studio Code (рис. 3**Error! No text of specified style in document.**..39).

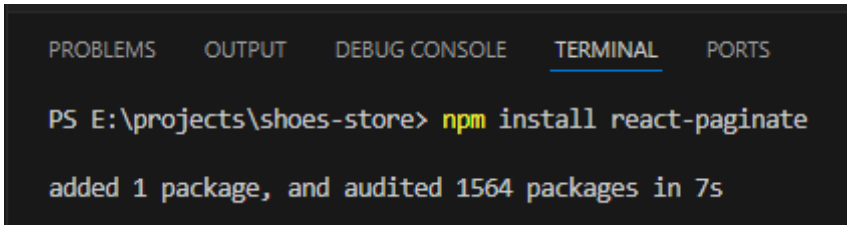


Рисунок 3**Error! No text of specified style in document.**..39 – Встановлення бібліотеки React Paginate

Бібліотека готова до налаштування і використання (рис. 3**Error! No text of specified style in document.**..40):

```
5  const Pagination = ({ currentPage, onChangePage }) => {
6    return (
7      <ReactPaginate
8        className={styles.root}
9        breakLabel="..."
10       nextLabel=">"
11       onPageChange={(event) => onChangePage(event.selected + 1)}
12       pageRangeDisplayed={8}
13       pageCount={3}
14       forcePage={currentPage - 1}
15       previousLabel="<"
16       renderOnZeroPageCount={null}
17     />
18   );
19 };
```

Рисунок 3**Error! No text of specified style in document.**..40 – Код налаштування компоненту пагінації

Розглянемо ключові частини:

- breakLabel="...", nextLabel=">", previousLabel="<" – ці властивості встановлюють текстові мітки для кнопок пагінації, де "..." – роздільник між сторінками, ">", "<" – кнопки "Наступна сторінка" та "Попередня сторінка" відповідно.

- onPageChange={(event) => onChangePage(event.selected + 1)} – Ця функція викликається, коли користувач змінює сторінку. Вона викликає функцію onChangePage, яку отримує компонент пагінації, передаючи індекс нової сторінки (+1, оскільки сторінки нумеруються з 1, а не з 0).
- pageRangeDisplayed={8} – властивість, яка вказує, скільки товарів відображати в межах однієї сторінки.
- pageCount={3} – кількість сторінок в пагінації.
- forcePage={currentPage - 1} – вказує на активну сторінку в пагінації (-1 від currentPage потрібно для відповідності нумерації сторінок, яка починається з 0).
- renderOnZeroPageCount={null} – Вказує, що компонент не має рендеритися, якщо кількість сторінок дорівнює 0.

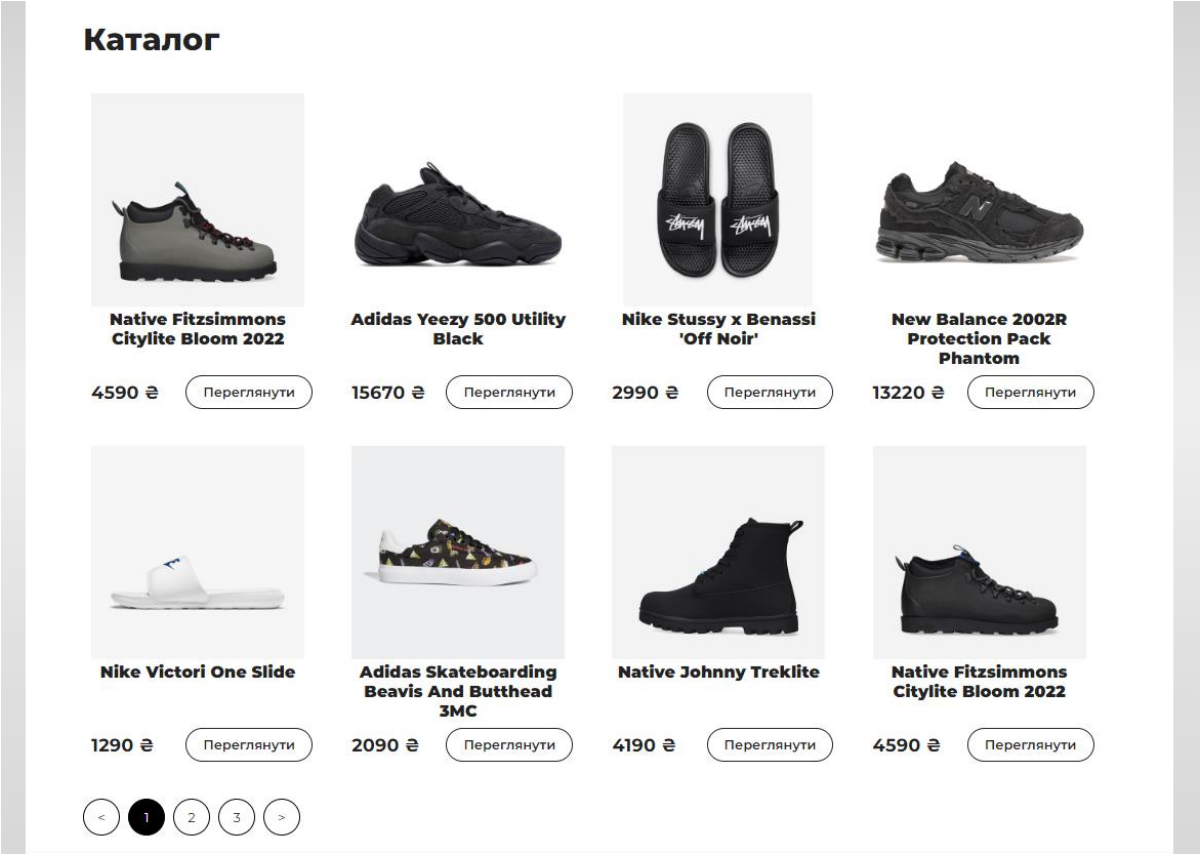


Рисунок 3**Error! No text of specified style in document.**.41 – Товари, що знаходятся на першій сторінці

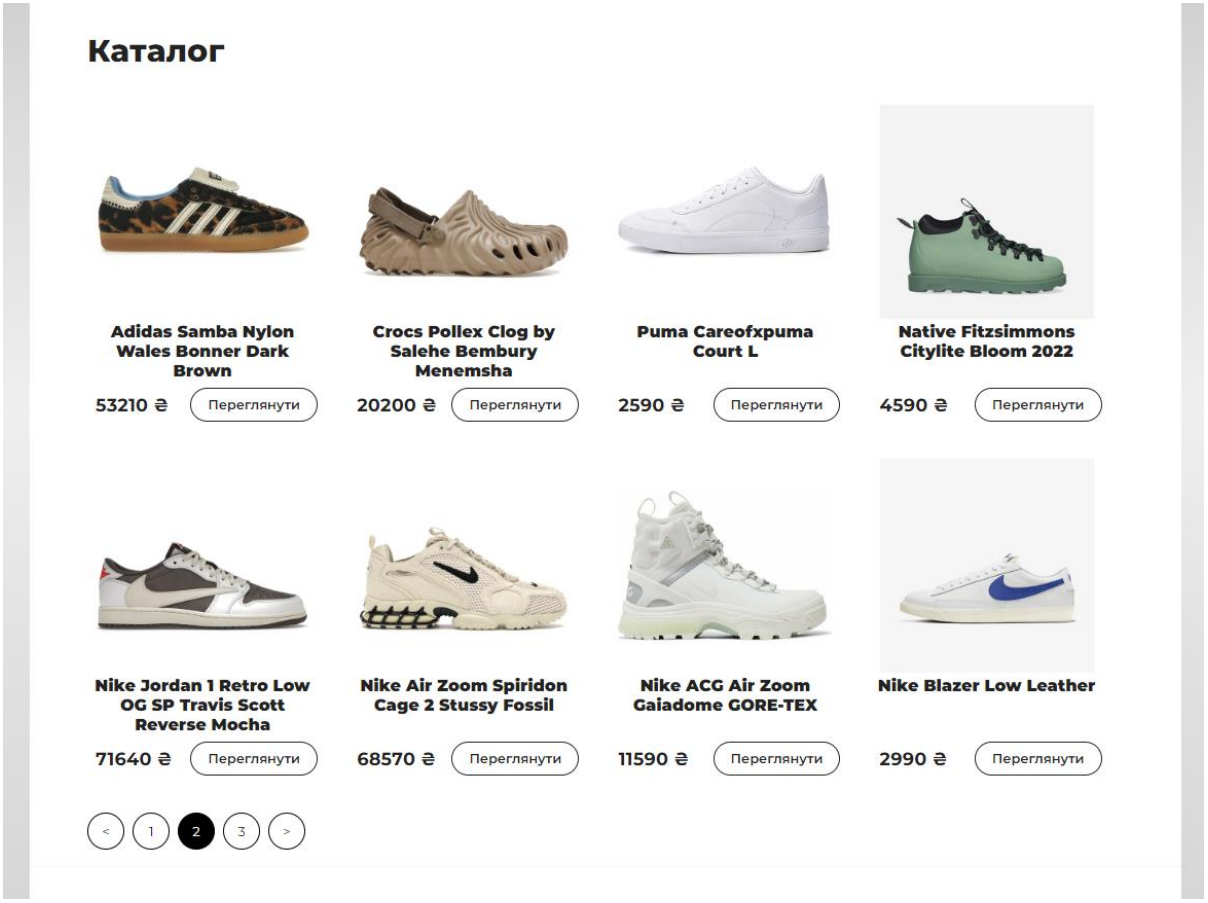


Рисунок 3**Error! No text of specified style in document.**.42 – Товари, що знаходятся на другій сторінці

Як можна побачити, пагінація працює.

Повний код файлу /Pagination/index.jsx наведено в **додатку Ї**.

3.4.3.4. Компонент категорій

Розглянемо реалізацію цього компонента (рис. 3.43):

```
import React from 'react';

function Categories({ value, onChangeCategory }) {
  const categories = ['Всі', 'Кросівки', 'Кеди', 'Черевики', 'Шльопанці'];

  return (
    <div className="categories">
      <ul>
        {categories.map((categoryName, i) => (
          <li key={i} onClick={() => onChangeCategory(i)} className={value === i ? 'active' : ''}>
            {categoryName}
          </li>
        ))}
      </ul>
    </div>
  );
}

export default Categories;

```

Рисунок 3.43 – Код розмітки компонента категорій

Розглянемо ключові частини:

- const categories = ['Всі', 'Кросівки', 'Кеди', 'Черевики', 'Шльопанці'] – цей масив містить список усіх доступних категорій товарів.
- – в цьому блоці буде відображено всі доступні категорії.
- – використовується для відображення списку категорій у вигляді ненумерованого списку.
- {categories.map((categoryName, i) => (...))} – метод map використовується для перетворення кожного елемента масиву categories у відповідний елемент JSX. Кожний товар, що зберігається у бекенді має свій номер категорії, що відповідає індексу масива categories.
- <li key={i} onClick={() => onChangeCategory(i)} className={value === i ? 'active' : ''}> – кожному елементу <li> присвоюється ключ i, що дозволяє React відслідковувати їхні зміни. Крім того, для кожної категорії встановлюється обробник події onClick, який викликає функцію onChangeCategory з індексом категорії. Далі йде умова, що визначає назву класу стилізації, і якщо індекс категорії, який зберігає слайс фільтрації співпадає з індексом вибраної категорії, тобто користувач обрав саме її, тоді

її необхідно виділити на фоні інших, для покращення користувацького досвіду.

- {categoryName} – назва категорії, отримана з масиву categories, відповідно до свого індексу.

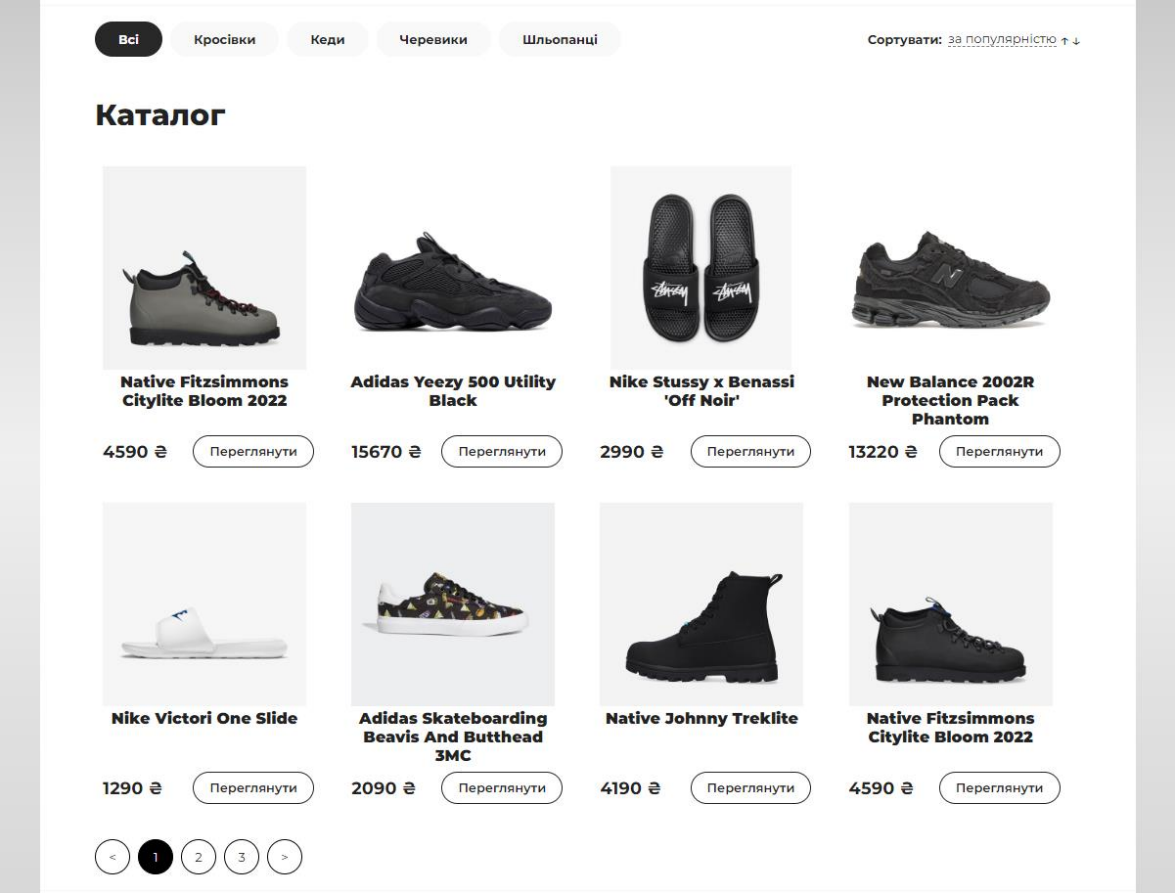


Рисунок 3.44 – Вигляд всіх товарів (без вибраної категорії)

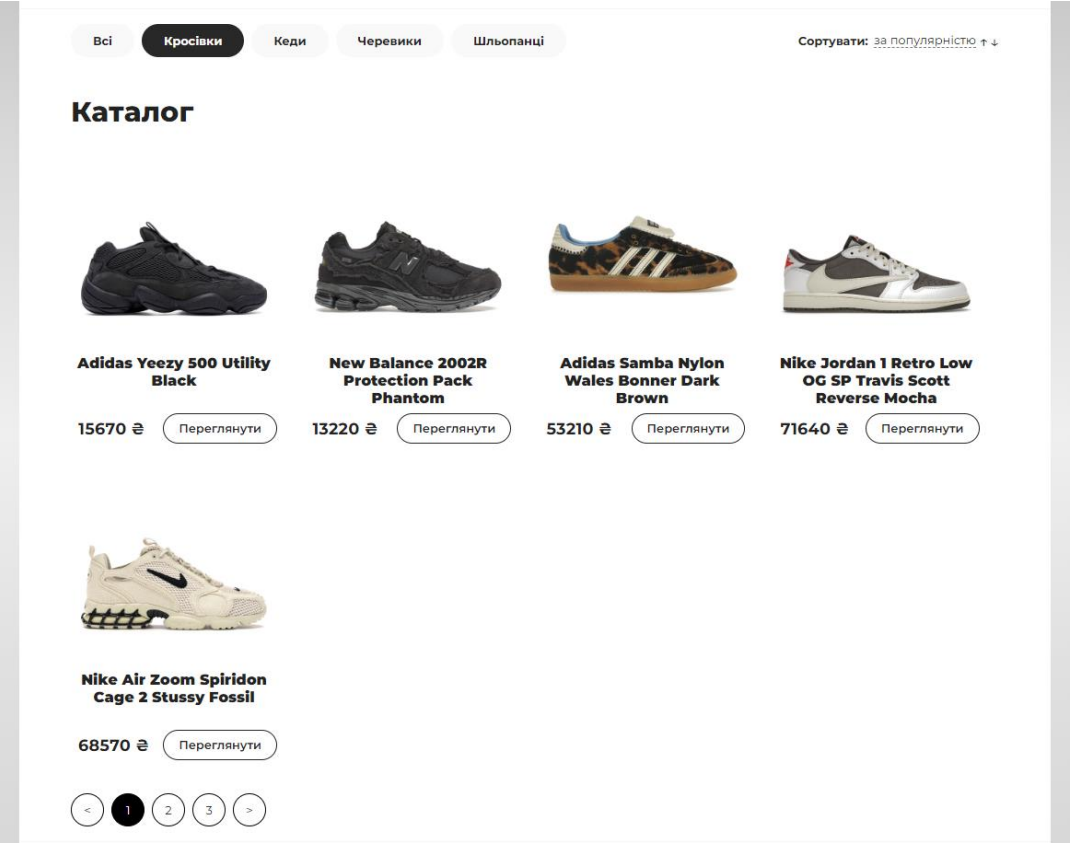


Рисунок 3.45 – Вигляд товарів категорії "Кросівки"

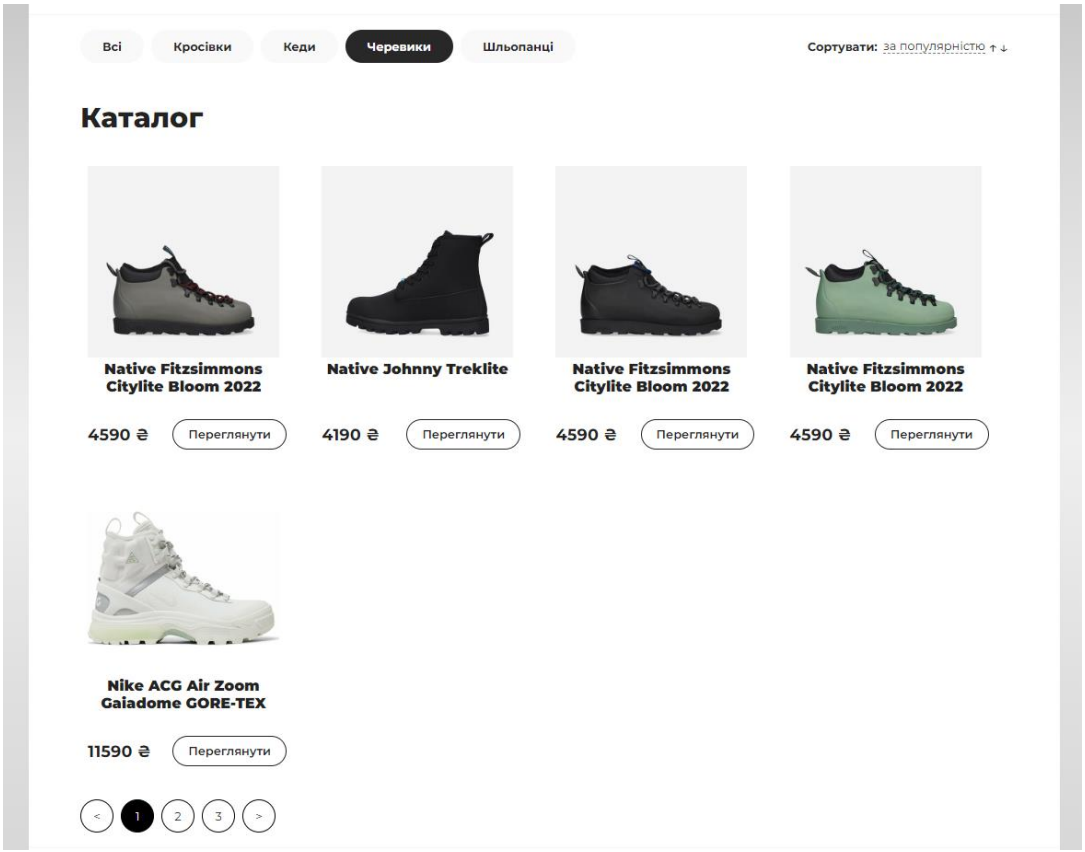


Рисунок 3.46 – Вигляд товарів категорії "Черевики"

Як можна побачити, компонент категорій реалізовано, і веб-сайт показує саме ті товари, категорії якого обрав користувач.

Повний код файлу Categories.jsx наведено в **додатку Й**.

3.4.3.5. Компонент сортування

Розглянемо реалізацію цього компонента (рис. 3**Error! No text of specified style in document.**.47):

```
return (
  <div ref={sortRef} className="sort">
    <div className="sort__label">
      <b>Сортувати:</b>
      <span onClick={() => setOpen(!open)}>{sort.name}</span>
      <button onClick={() => setOrder('asc')}>↑</button>
      <button onClick={() => setOrder('desc')}>↓</button>
    </div>
    {open && (
      <div className="sort__popup">
        <ul>
          {list.map((obj, i) => (
            <li
              key={i}
              onClick={() => onClickListItem(obj)}
              className={sort.sortProperty === obj.sortProperty ? 'active' : ''}>
              {obj.name}
            </li>
          ))}
        </ul>
      </div>
    )}
  </div>
);
```

Рисунок 3**Error! No text of specified style in document.**.47 – Код розмітки компонента сортування товарів

Розглянемо ключові частини:

- – в межах цього блоку розташовується мітка сортування, кнопки порядку сортування, та випадаюче вікно з параметрами сортування
- – в цьому блоці відображається мітка "Сортувати:", яка є статичним текстом, поряд з нею виводиться поточний вибраний тип сортування, який отримується зі слайсу filter. Коли користувач клікає на цей текст, відбувається зміна стану open, що призводить до відкриття спливаючого вікна сортування. Також, є кнопки зі стрілками вгору та вниз, за допомогою яких можна вибрати порядок сортування (зростання або спадання).
- {open && (…} – якщо стан open зберігає в собі значення true, то відображається випадаюче вікно сортування, де

користувач може вибрати параметри сортування. Для того, щоб закрити це вікно, користувач має натиснути за межі блоку , отже потрібно відстежити клік користувача за межами блоку, для цього і передбачена змінна sortRef (рис. 3**Error! No text of specified style in document.**.48):

```
React.useEffect(() => {
  const handleClickOutside = (event) => {
    if (!event.composedPath().includes(sortRef.current)) {
      setOpen(false);
    }
  };
```

Рисунок 3**Error! No text of specified style in document.**.48 – Код функції для відстеження кліків користувача поза межами блоку сортування

У випадку, якщо користувач клікне за межами блоку сортування, стан змінної open зміниться на false, і відповідно рендер цього вікна припиниться.

- onClick={() => onClickListItem(obj)} – функція, для зміни параметрів сортування (рис. 3**Error! No text of specified style in document.**.49):

```
const onClickListItem = (obj) => {
  dispatch(setSort(obj));
  setOpen(false);
};
```

Рисунок 3**Error! No text of specified style in document.**.49 – Код функції onClickListItem

Функція onClickListItem() змінить стан слайсу фільтрації, а саме поля sort, що спричинить зміну get-запиту на бекенд і відобразить користувачеві товари з вибраними параметрами сортування, після цього функція закриє спливаюче вікно.

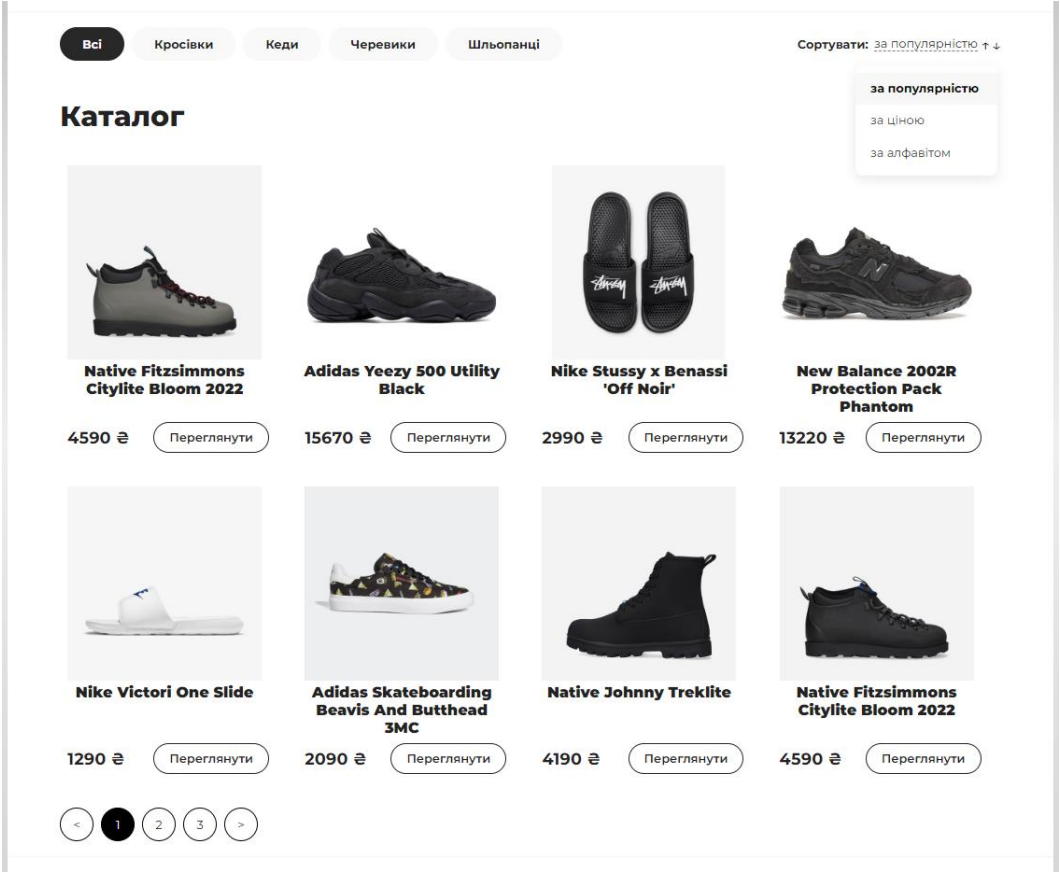


Рисунок 3**Error! No text of specified style in document.**.50 – Сортування товарів за популярністю

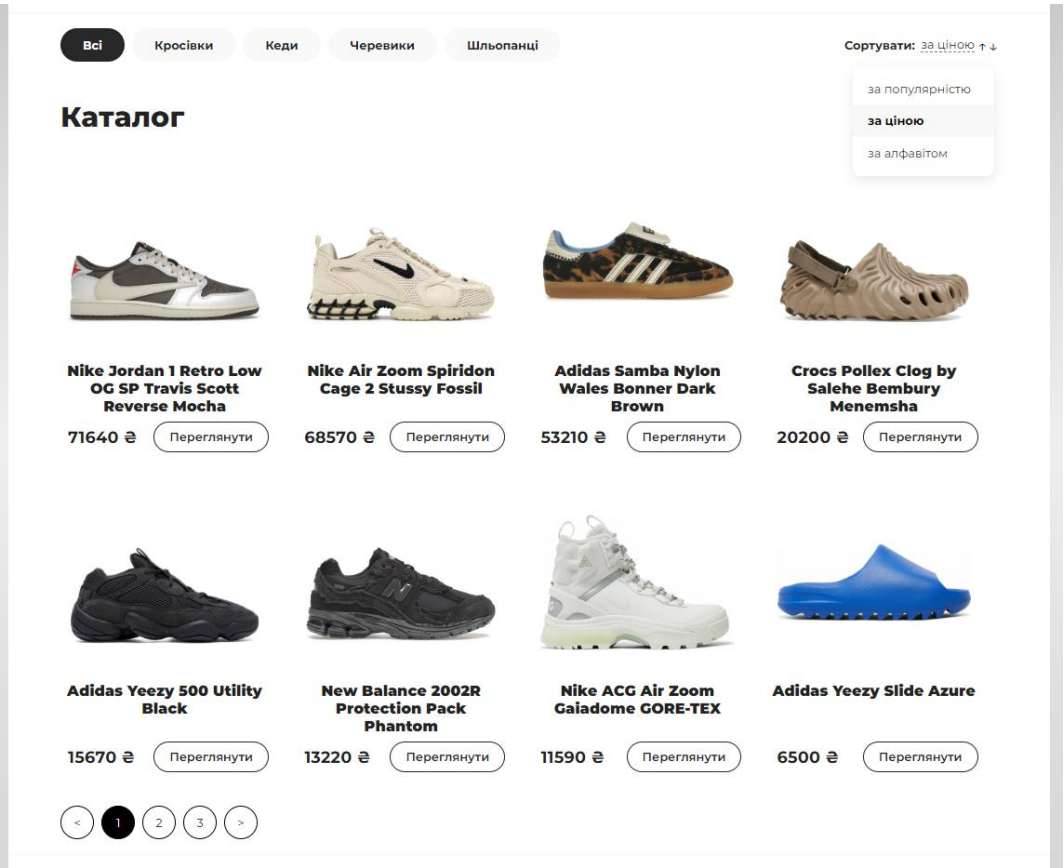


Рисунок 3**Error! No text of specified style in document.**.51 – Сортування товарів за ціною

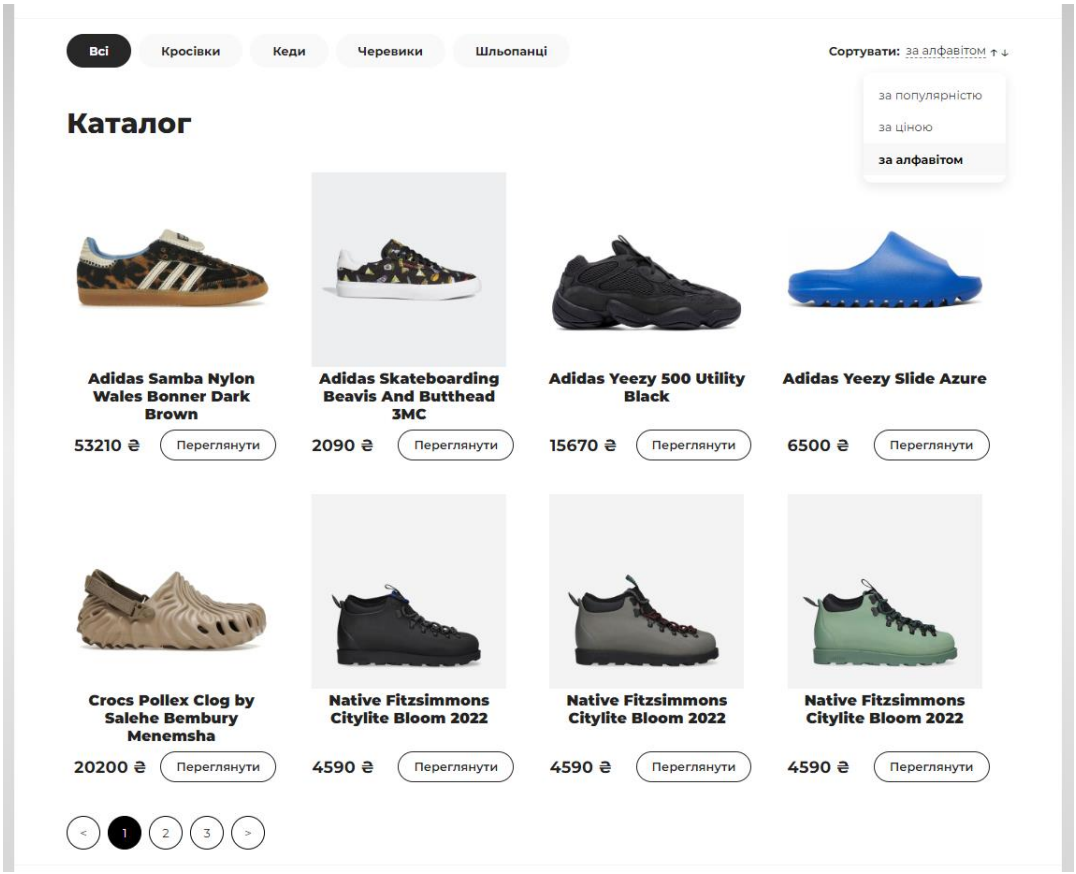


Рисунок 3**Error! No text of specified style in document.**.52 – Сортування товарів за алфавітом

Таким чином, реалізовано функціонал сортування товарів за різними параметрами за спаданням або за зростанням.

Повний код файлу Sort/index.jsx наведено в **додатку К**.

3.4.4. Сторінка товару

На головній сторінці, натиснувши в межах блоку товару, що цікавить користувача, він потрапляє на сторінку товару, де може побачити його опис, вибрати розмір і додати у кошик. Розглянемо розмітку цієї сторінки (рис. 3.53):

```
10  const Product = () => {
62    return (
63      <div className="container__product">
64        <div className="image-container">
65          <img src={shoes.imageUrl} alt="Зображення товару" />
66        </div>
67        <div className="info-container">
68          <h1>{shoes.title}</h1>
69          <h2>Ціна: {shoes.price} ₴</h2>
70          <h3>Оберіть розмір:</h3>
71          <div className="shoes-block">
72            <div className="size-selector">
73              <div className="shoes-block__selector">
74                <ul>
75                  {shoes.sizes &&
76                    shoes.sizes.map((size, i) => (
77                      <li
78                        key={size}
79                        onClick={() => setActiveSize(i)}
80                        className={activeSize === i ? 'active' : ''}>
81                        {size}
82                      </li>
83                    ))}
84                </ul>
85              </div>
86              <div className="shoes-block__bottom">
87                <button
88                  onClick={() => onClickAdd(shoes)}
89                  className="button button--outline button--add">
90 ›                <svg ···
100               </svg>
101               <span>В кошик</span>
102             </button>
103           </div>
104         </div>
105       </div>
106       <h3>Опис:</h3>
107       <p>{shoes.description}</p>
108     </div>
109   </div>
```

Рисунок **Error! No text of specified style in document.**3.53 – Код розмітки сторінки товару

Розглянемо ключові частини:

- – контейнер, що визначає область для відображення товару.
- – в цьому блоці відображається зображення товару.
- – зображення товару зчитується з властивості imageUrl об'єкта shoes і відображається на сторінці.
- – цей блок містить інформацію про товар.
- <h1>{shoes.title}</h1> – відображає назву товару, яка зчитується з властивості title об'єкта shoes.
- <h2>Ціна: {shoes.price} ₴</h2> – відображає ціну товару, яка зчитується з властивості price об'єкта shoes.
- <h3>Оберіть розмір:</h3> – підказка користувачеві про те, що він може обрати розмір товару.
- {shoes.sizes && ...} – перевірка, чи існують доступні розміри (sizes) для товару. Якщо так, відображається список розмірів.
- {shoes.sizes.map((size, i) => (...)} – використовується метод map для відображення кожного розміру як елемент списку <li>.
- onClick={() => setActiveSize(i)} – при кліку на один з доступних розмірів змінюється стан вибраного розміру за допомогою функції setActiveSize, що в подальшому забезпечіть можливість зберегти в кошику саме той розмір взуття, який обрав користувач.
- <button onClick={() => onClickAdd(shoes)} ... – кнопка "В кошик", яка додає товар до кошика. При кліку викликається функція onClickAdd, яка додає обраний товар до кошика (рис. 3.54):

```
50  const onClickAdd = () => {
51    const { id, title, price, imageUrl, sizes } = shoes;
52    const item = {
53      id,
54      title,
55      price,
56      imageUrl,
57      size: sizes[activeSize],
58    };
59    dispatch(addItem(item));
60  };
```

Рисунок **Error! No text of specified style in document.**3.54 – Код функції onClickAdd

Функція onClickAdd() передає у слайс кошика (3.3.2 Стан кошика) повну інформацію про товар і обраний користувачем розмір.

- <h3>Опис:</h3> – Підказка про те, що нижче буде відображено опис товару.
- {shoes.description} – Відображає опис товару, який зчитується з властивості description об'єкта shoes.

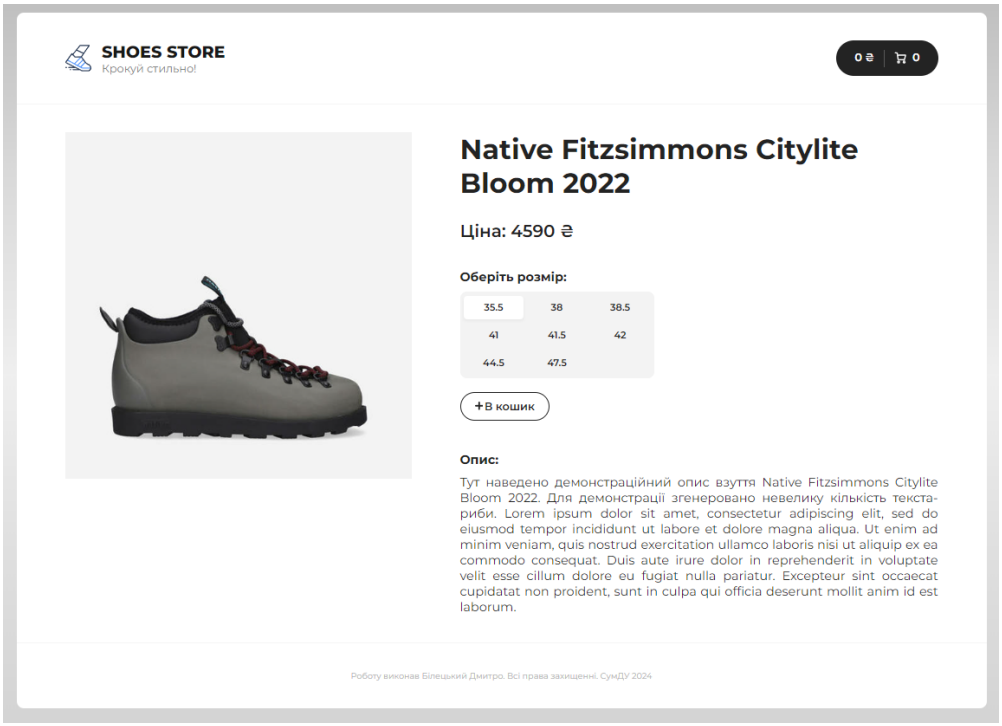


Рисунок **Error! No text of specified style in document.**3.55 – Вигляд сторінки товару

Таким чином, реалізовано сторінку товару, де можна побачити його опис, вибрати розмір і додати у кошик.

Повний код файлу Product.jsx наведено в **додатку Л**.

3.4.5. Сторінка кошика

Якщо користувач перейде на сторінку кошика, не додавши туди жодного товару, він побачить відповідне повідомлення (рис. 3**Error! No text of specified style in document.**.56):

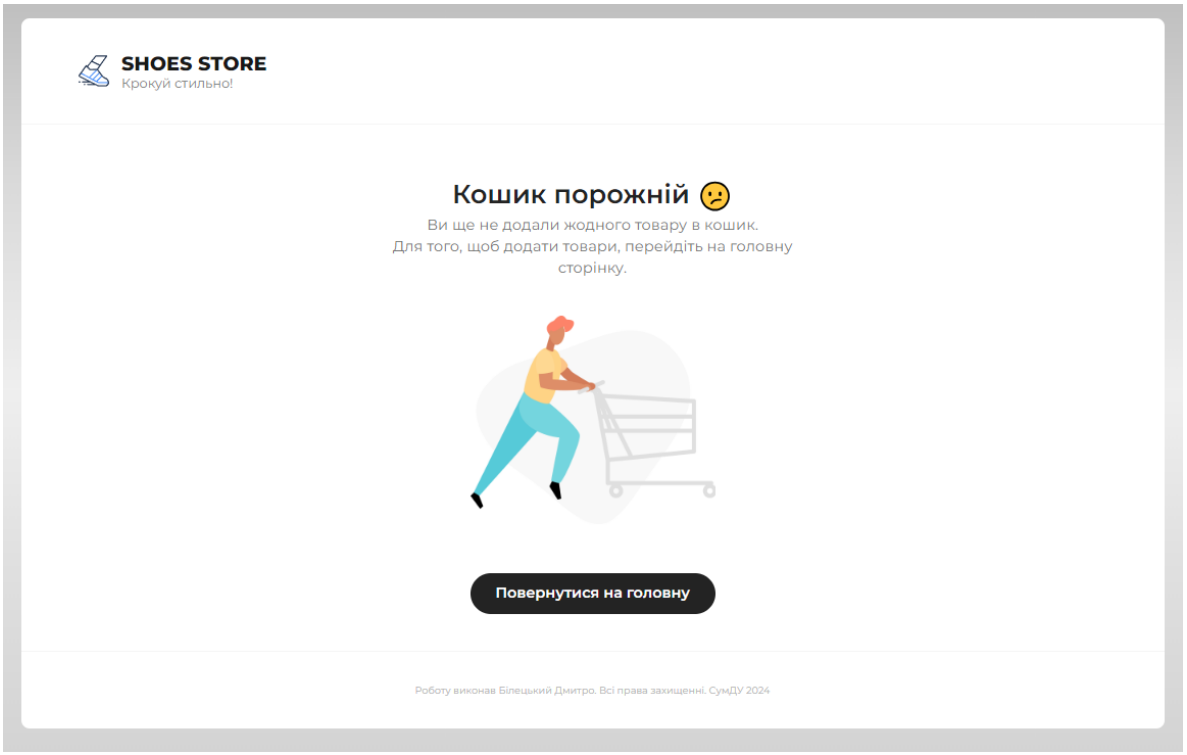


Рисунок 3**Error! No text of specified style in document.**.56 – Вигляд сторінки порожнього кошика

Повний код файлу CartEmpty.jsx наведено в **додатку М**.

У випадку, якщо користувач додав хоча б один товар до кошика, його зустріне сторінка, де він зможе побачити товари, які обрав, матиме змогу збільшити чи зменшити їх кількість відповідними кнопками, видалити їх з кошика по одному чи очистити весь кошик відразу. Розглянемо розмітку цієї сторінки (рис. 3**Error! No text of specified style in document.**.57):

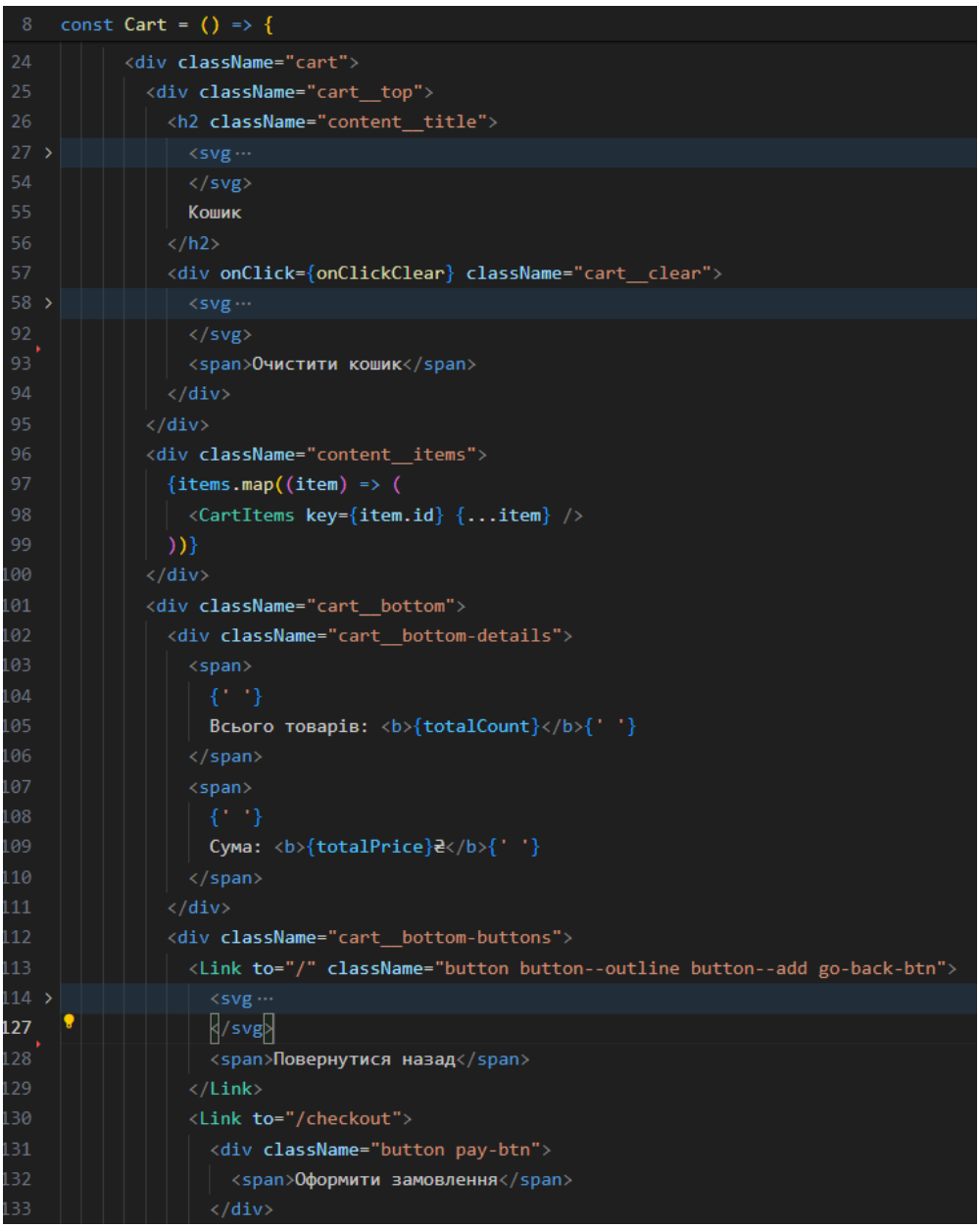


Рисунок 3**Error! No text of specified style in document.**.57 – Код розмітки сторінки кошика товарів

Розглянемо ключові частини:

- – блок містить всі товари, які додано до кошика.
- –блок містить заголовок сторінки "Кошик" та кнопку "Очистити кошик".
- <h2 className="content__title">...</h2> – відображає заголовок "Кошик" із зображенням кошика.

- – кнопка "Очистити кошик", яка викликає функцію onClickClear при кліку (рис. 3**Error! No text of specified style in document.**.58):

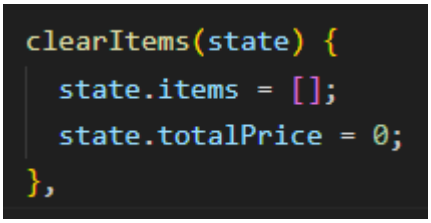


Рисунок 3**Error! No text of specified style in document.**.58 – Код функції для очищення кошика

Функція clearItems() замість товарів, що зберігає слайс кошика, записує пустий масив і обнуляє загальну вартість товарів.

- {items.map((item) => (<CartItems key={item.id} {...item} />))} – відображає кожен товар у кошику. Кожен товар рендериться за допомогою компонента CartItems з використанням методу map.
- – блок містить загальну кількість товарів і загальну суму, а також кнопки "Повернутися назад" та "Оформити замовлення".
- <Link to="/" className="button button--outline button--add go-back-btn">...</Link> – кнопка "Повернутися назад", яка перенаправляє користувача на головну сторінку.
- <Link to="/checkout">...</Link> – кнопка "Оформити замовлення", яка перенаправляє користувача на сторінку оформлення замовлення.

Розглянемо розмітку компонента <CartItems /> (рис. 3**Error! No text of specified style in document.**.59):

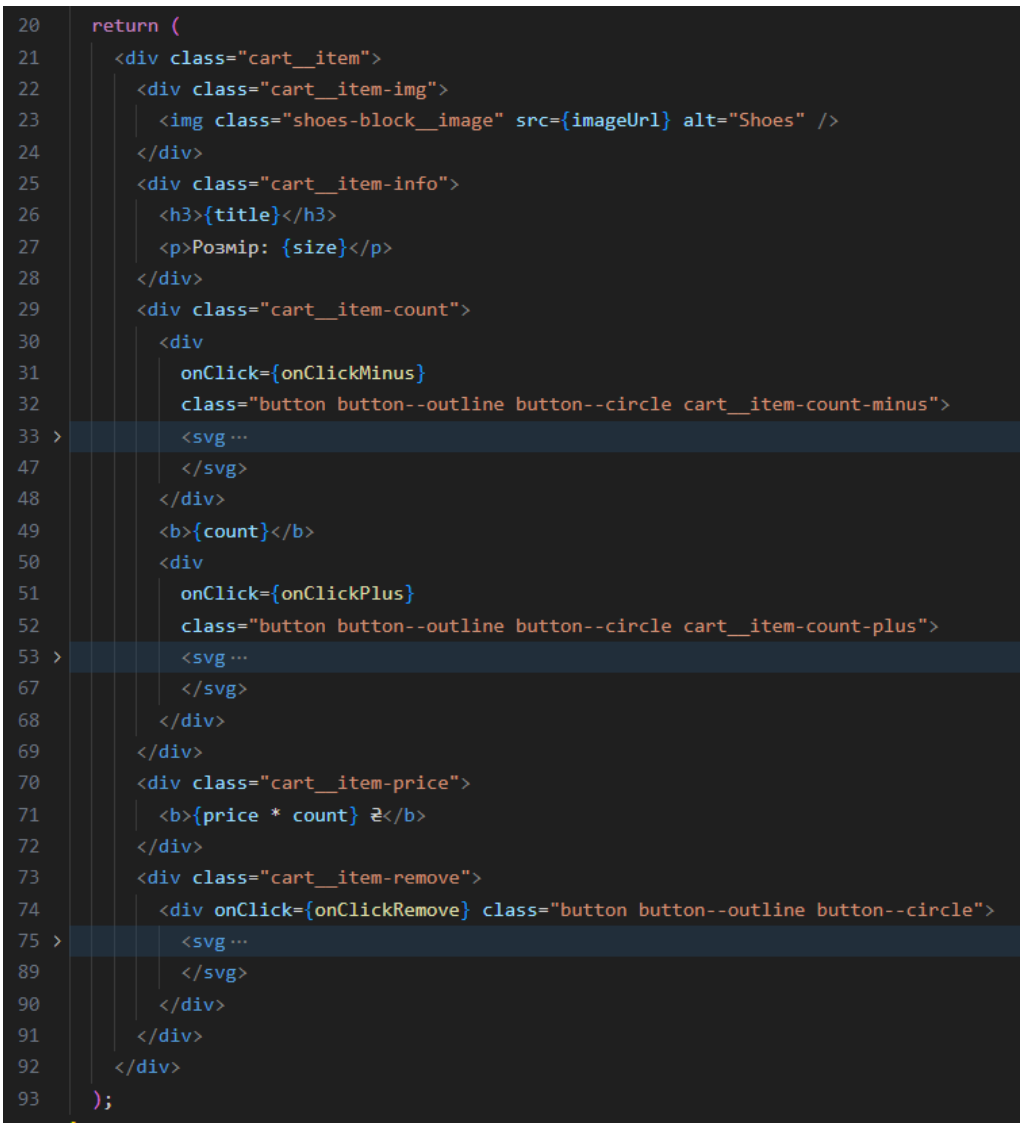


Рисунок 3**Error! No text of specified style in document.**.59 – Код розмітки компонента CartItem

Розглянемо ключові частини:

- – основний контейнер для одного товару у кошику.
- – контейнер для зображення товару.
- – зображення товару.
- – контейнер для інформації про товар.
- <h3>{title}</h3> – назва товару.
- Розмір: {size} – розмір товару.
- – контейнер для керування кількістю товару.

- – кнопка "Мінус", яка віднімає одиницю від кількості товару (рис. 3**Error! No text of specified style in document.**.60):

```
minusItem(state, action) {
  const findItem = state.items.find((obj) => {
    return obj.id === action.payload.id && obj.size === action.payload.size;
  });
  console.log(findItem);
  if (findItem && findItem.count > 1) {
    findItem.count--;
    state.totalPrice = state.totalPrice - findItem.price;
  }
},
```

Рисунок 3**Error! No text of specified style in document.**.60 – Код функції для зменшення кількості товарів кошика

Спочатку функція minusItem() знаходить потрібний товар серед інших, важливо шукати товар не тільки за його id, а ще і за розміром взуття, який обрав користувач, оскільки в корзині може бути одна і та сама модель взуття, але з різними розмірами. Після того як товар знайдено, необхідно зменшити його кількість на 1 і перерахувати загальну вартість. Крім того, було створено перевірку, яка не дасть викликати функцію для товару, кількість якого дорівнює 1. Якщо користувач захоче видалити конкретну модель з конкретним розміром, він скористається відповідною кнопкою.

- {count} – відображає поточну кількість конкретного товару.
- – Кнопка "Плюс", яка додає одиницю до кількості товару. Для реалізації функціоналу викликається та сама функція, що і для додавання товару до кошику.
- {price * count} ₴ – відображає загальну ціну конкретної моделі з конкретним розміром, враховуючи її кількість у кошику.
- – кнопка "Видалити", яка видаляє товар з кошика (рис. 3**Error! No text of specified style in document.**.61):

```
removeItem(state, action) {
  state.items = state.items.filter(
    (obj) => !(obj.id === action.payload.id && obj.size === action.payload.size),
  );
  state.totalPrice = calcTotalPrice(state.items);
},
```

Рисунок 3**Error! No text of specified style in document.**.61 – Код функції для видалення товару з кошика

Функція removeItem() спочатку знаходить необхідний товар, враховуючи обраний користувачем розмір, потім фільтрує всі товари кошику без врахування товару, що видаляється і після цього перераховує загальну вартість товарів в кошику.

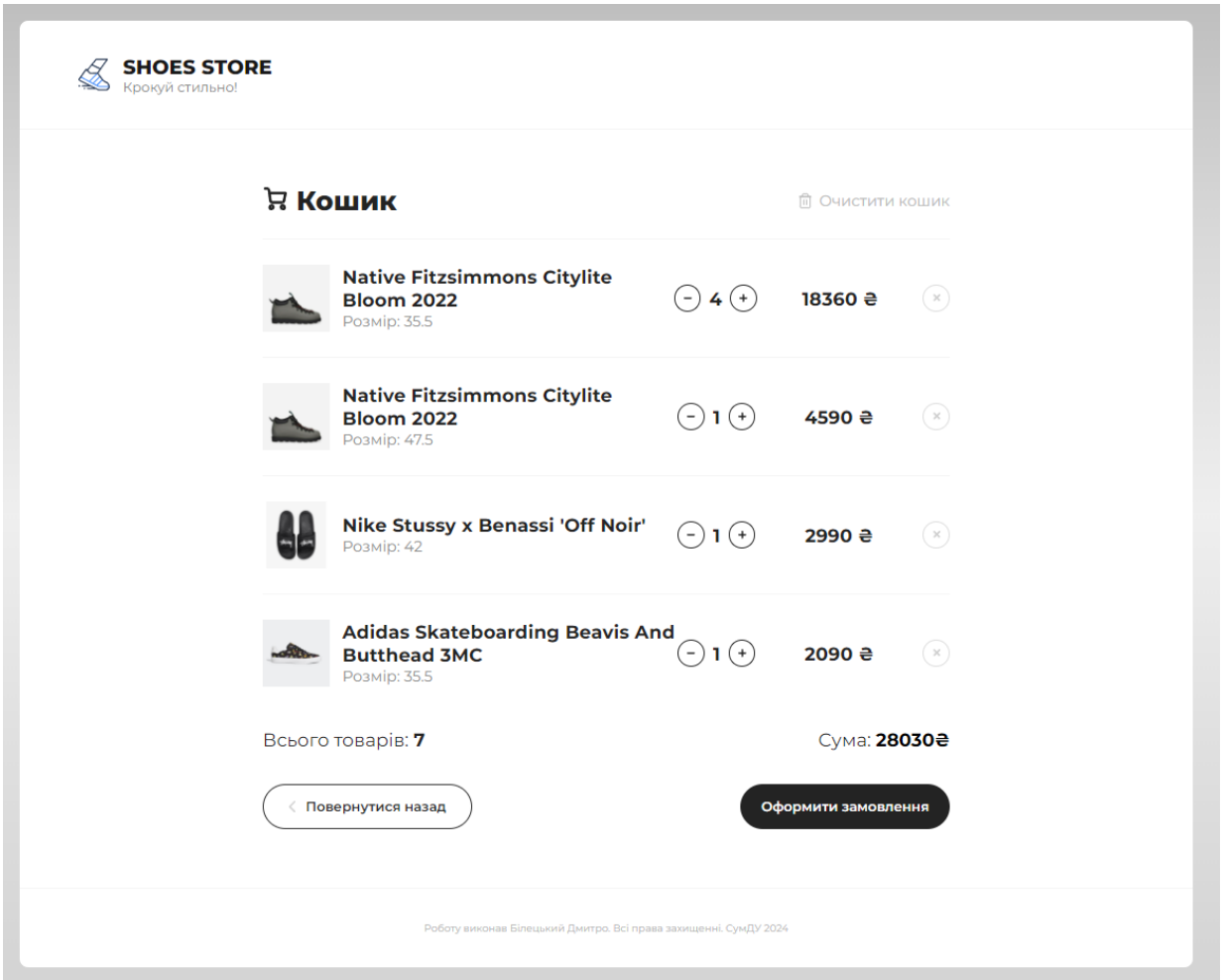


Рисунок **Error! No text of specified style in document.**3.62 – Вигляд сторінки кошика товарів

Таким чином, реалізовано функціонал кошика.

Повний код файлу Cart.jsx наведено в **додатку Н**.

Повний код файлу CartItems.jsx наведено в **додатку О**.

3.4.6. Сторінка 404

При спробі перейти на сторінку, що не існує в межах веб-сайту, що реалізується, користувач потрапить на так звану сторінку 404, де побачить інформаційне повідомлення про відсутність сторінки (рис. 3**Error! No text of specified style in document.**.63):

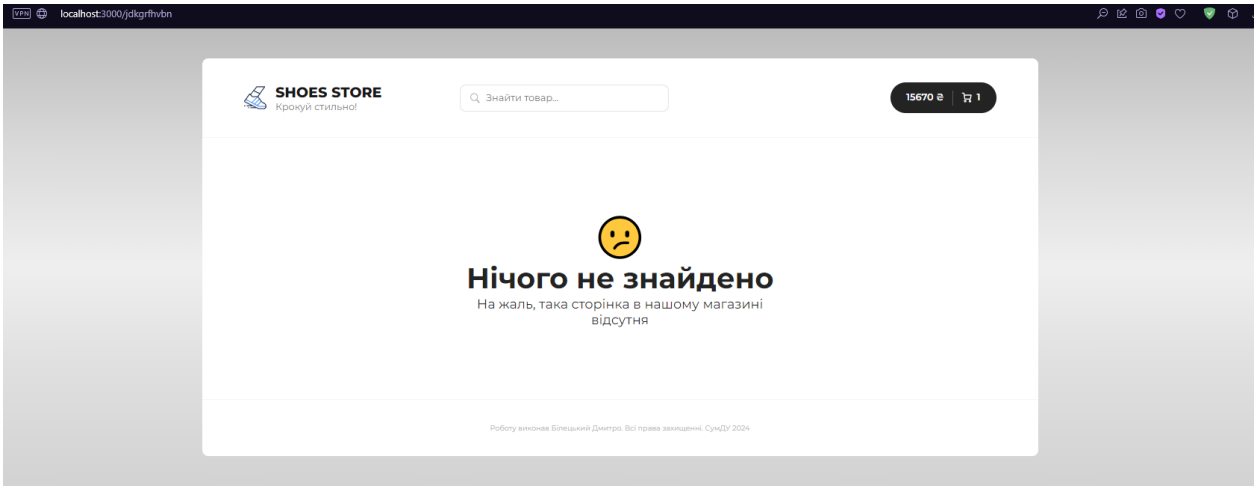


Рисунок 3**Error! No text of specified style in document.**.63 – Вигляд сторінки 404

Повний код файлу /NotFoundBlock/index.jsx наведено в **додатку П**.

Повний код файлу NotFound.jsx наведено в **додатку Р**.

3.4.7. Сторінка оформлення замовлення

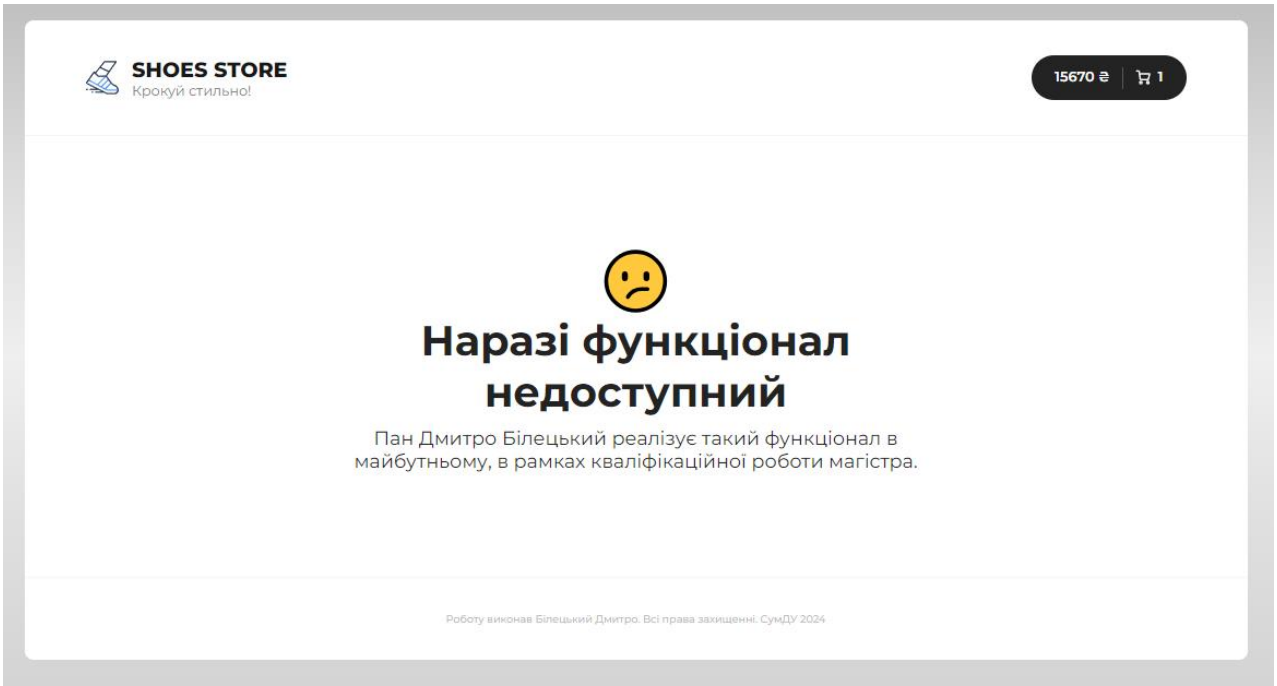


Рисунок **Error! No text of specified style in document.**3.64 – Вигляд сторінки оформлення замовлення

Функціонал оформлення замовлення потребує створення власного бекенду та панелі адміністрування. В рамках кваліфікаційної роботи бакалавра було повністю реалізовано фронт-енд частину додатку, і продемонстровано можливість зв’язати її зі стороннім бекендом (MockAPI). Кваліфікаційна робота магістра ідейно продовжить розвиток веб-сайту для продажу взуття, забезпечивши можливість користувачам оформлювати замовлення.

3.4.8. Реалізація навігації між сторінками

Для реалізації функціоналу навігації було використано бібліотеку React Router. Для початку, її необхідно встановити у проект виконавши відповідний код в терміналі Visual Studio Code (рис. 3**Error! No text of specified style in document.**.65):

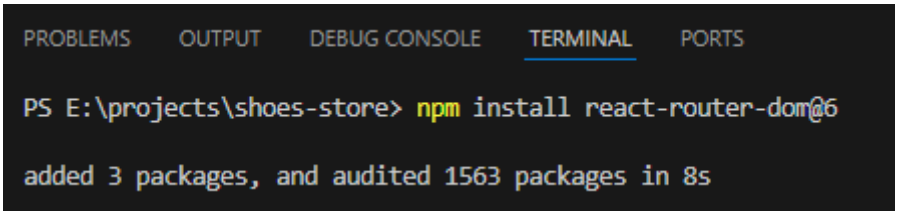


Рисунок 3**Error! No text of specified style in document.**.65 – Встановлення бібліотеки React Router

Бібліотека встановлена і готова до використання. Для початку, необхідно забезпечити рендер головного компоненту веб-сайту в межах компонента <BrowserRouter> (рис. 3**Error! No text of specified style in document.**.66):

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

Рисунок 3**Error! No text of specified style in document.**.66 – Підключення функціоналу навігації до веб-додатка

Таким чином, використання функціонала навігації тепер доступно для веб-сайту, що реалізується. Далі, налаштовано безпосередньо саму навігацію (рис. 3**Error! No text of specified style in document.**.67):

```
function App() {
  return (
    <Routes>
      <Route path="/" element={<MainLayout />}>
        <Route path="" element={<Home />} />
        <Route path="cart" element={<Cart />} />
        <Route path="shoes/:id" element={<Product />} />
        <Route path="checkout" element={<CheckOut />} />
        <Route path="*" element={<NotFound />} />
      </Route>
    </Routes>
  );
}
```

Рисунок 3**Error! No text of specified style in document.**.67 – Код налаштування навігації між сторінками веб-додатка

- <Routes> – компонент, який визначає контейнер для визначення маршрутів навігації
- <Route path="/"> – головний маршрут, який вказує на компонент <MainLayout />, що буде використовуватися для всіх сторінок веб-сайта (рис. 3.68):

Рисунок **Error! No text of specified style in document.**3.68 – Код розмітки компонента MainLayout

Header та Footer – невід'ємні частини кожної сторінки, це компоненти які будуть відображатися у будь-якому випадку. А компонент <Outlet /> визначає який саме контент сторінки буде рендеритися, враховуючи на якій саме сторінці користувач знаходиться.

- <Route path="" element={<Home />} /> – навігаційний маршрут для головної сторінки додатку, який відображає компонент <Home />.
- <Route path="cart" element={<Cart />} /> – Навігаційний маршрут для сторінки кошика, який відображає компонент <Cart />.
- <Route path="shoes/:id" element={<Product />} /> – навігаційний маршрут для сторінки окремого товару, який використовує динамічний параметр :id для ідентифікації конкретного товару.
- <Route path="checkout" element={<CheckOut />} /> – навігаційний маршрут для сторінки оформлення замовлення, який відображає компонент <CheckOut />.
- <Route path="*" element={<NotFound />} /> – навігаційний маршрут за замовчуванням, який відображається, якщо користувач введе неправильний URL (неіснуючу сторінку).

Повний код файлу App.js наведено в **додатку С**.

3.5. Забезпечення адаптивної верстки

Адаптивність веб-сайту забезпечена для наступних ширин екрану:

1. 360 пікселей
2. 393 пікселей
3. 412 пікселей
4. 768 пікселей
5. 800 пікселей
6. 1280 пікселей

Оскільки, саме вище перераховані ширини екрану є найпопулярнішими серед користувачів смартфонів та планшетів в Україні станом на квітень 2024 року [20-21].

3.5.1. Домашня сторінка

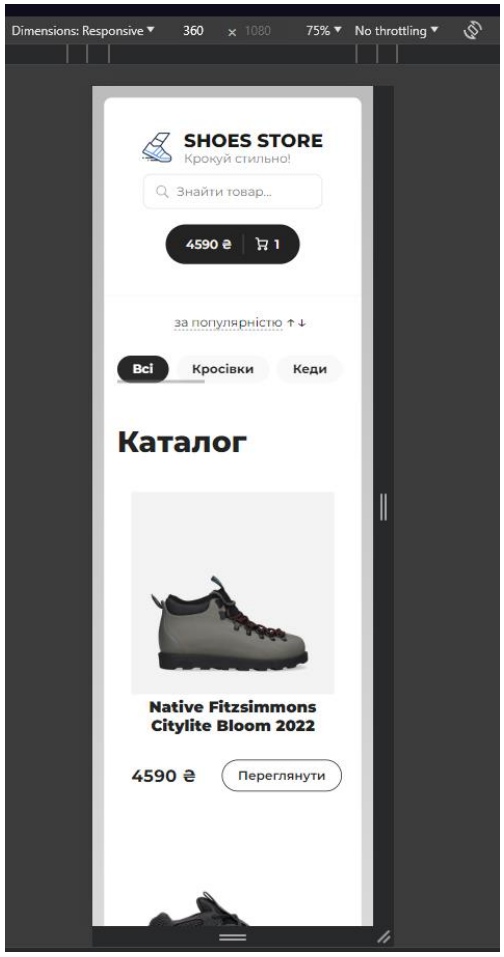


Рисунок **Error! No text of specified style in document.**3.69 – Вигляд домашньої сторінки при ширині екрану 360 пікселів

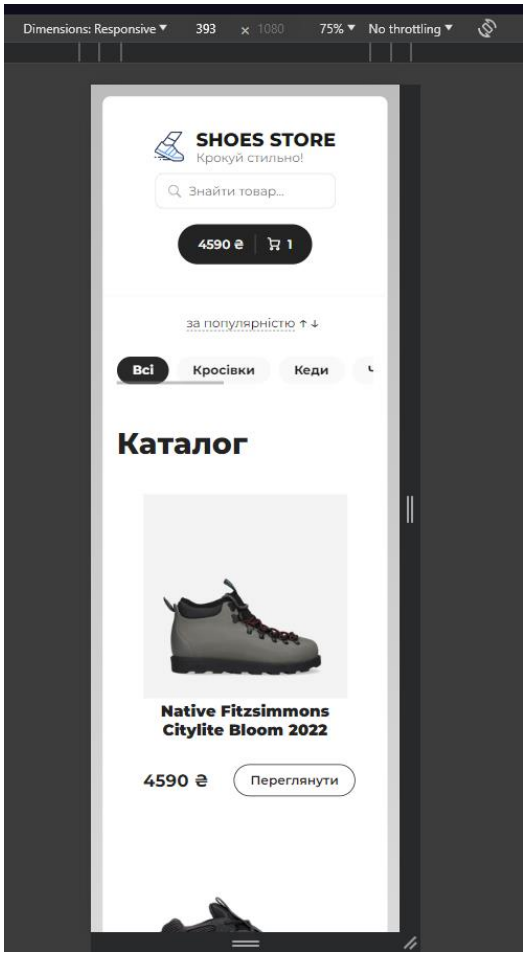


Рисунок 3.70 – Вигляд домашньої сторінки при ширині екрану 393 пікселів

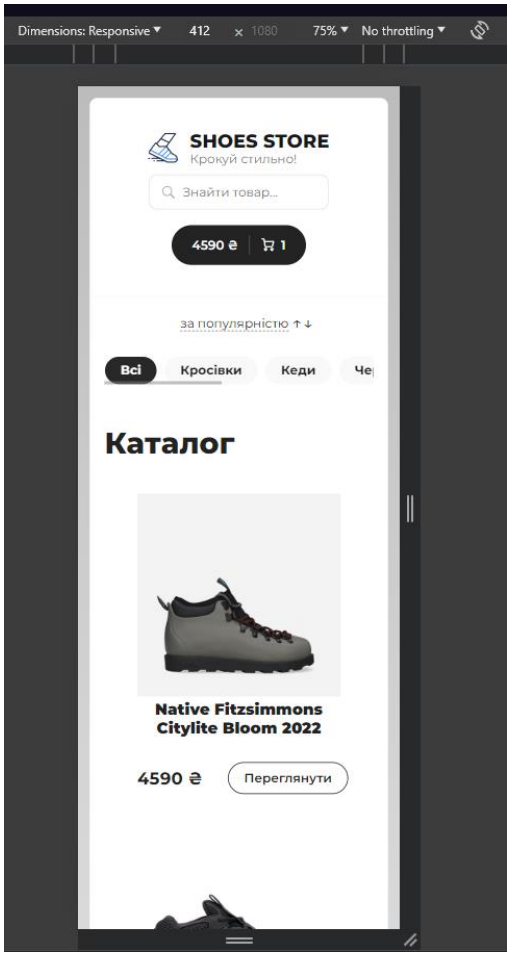


Рисунок 3.71 – Вигляд домашньої сторінки при ширині екрану 412 пікселів

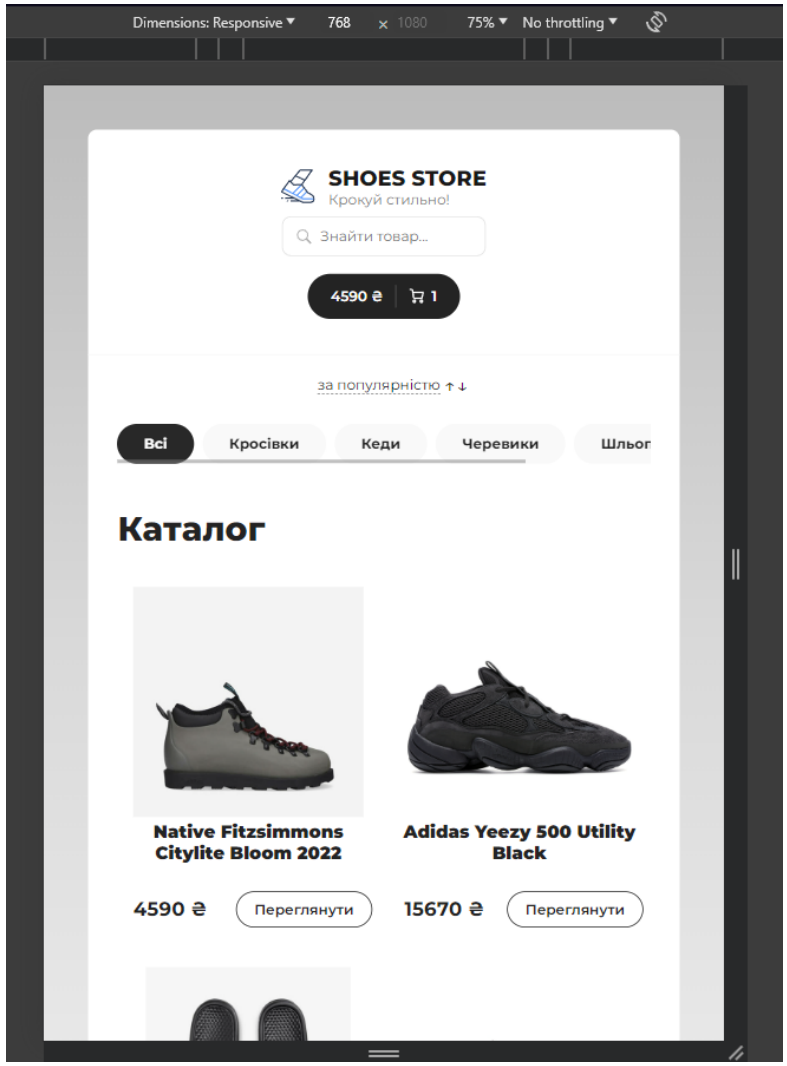


Рисунок **Error! No text of specified style in document.**3.72 – Вигляд домашньої сторінки при ширині екрану 768 пікселів

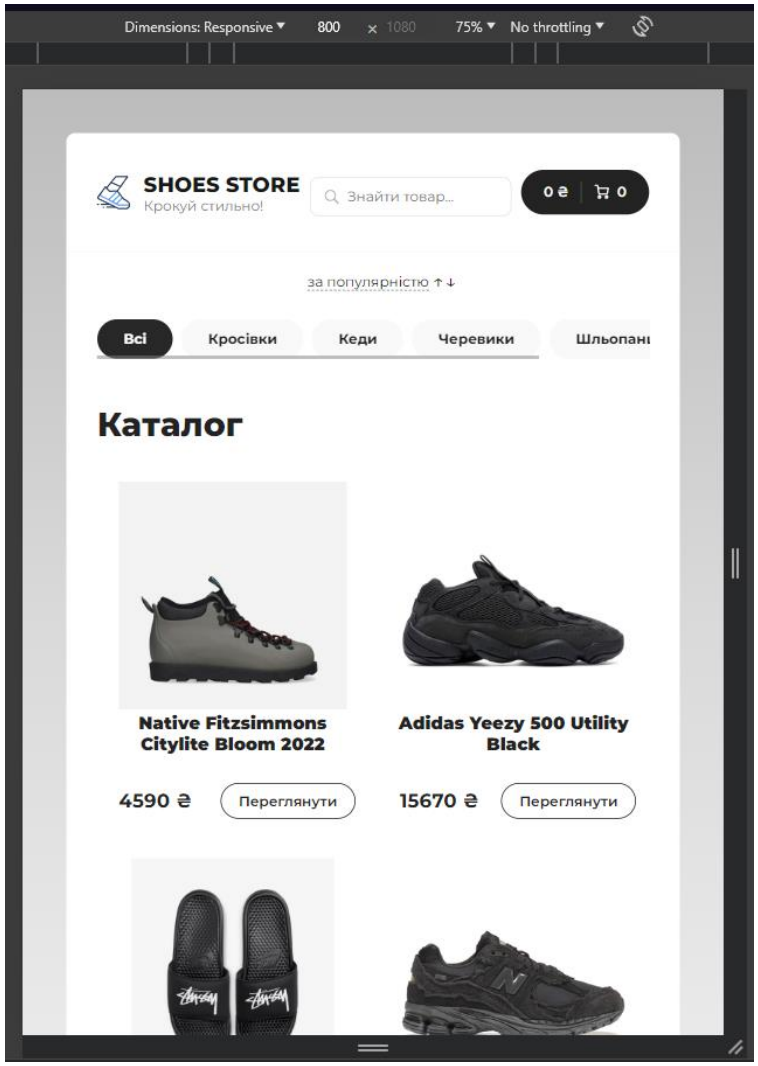


Рисунок **Error! No text of specified style in document.**3.73 – Вигляд домашньої сторінки при ширині екрану 800 пікселів

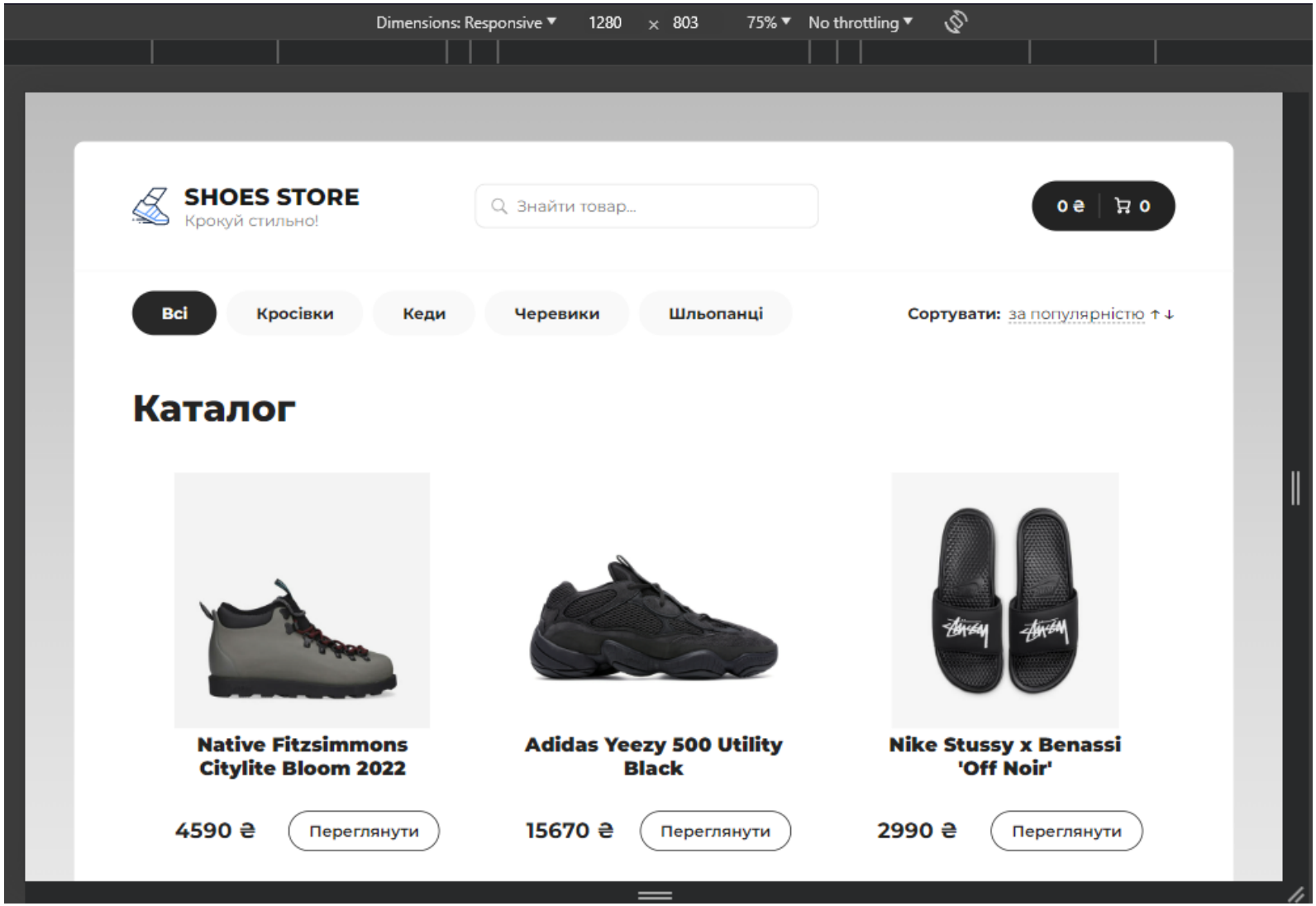


Рисунок **Error! No text of specified style in document.**3.74 – Вигляд домашньої сторінки при ширині екрану 1280 пікселів

Повний код стилізації наведено в git-репозитарії [19].

3.5.2. Сторінка товару

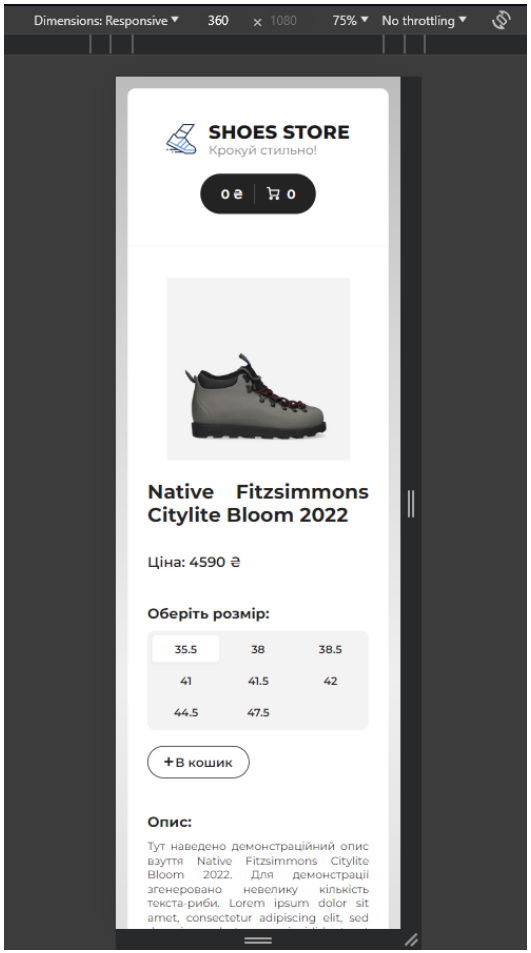


Рисунок 3.75 – Вигляд сторінки товару при ширині екрану 360 пікселів

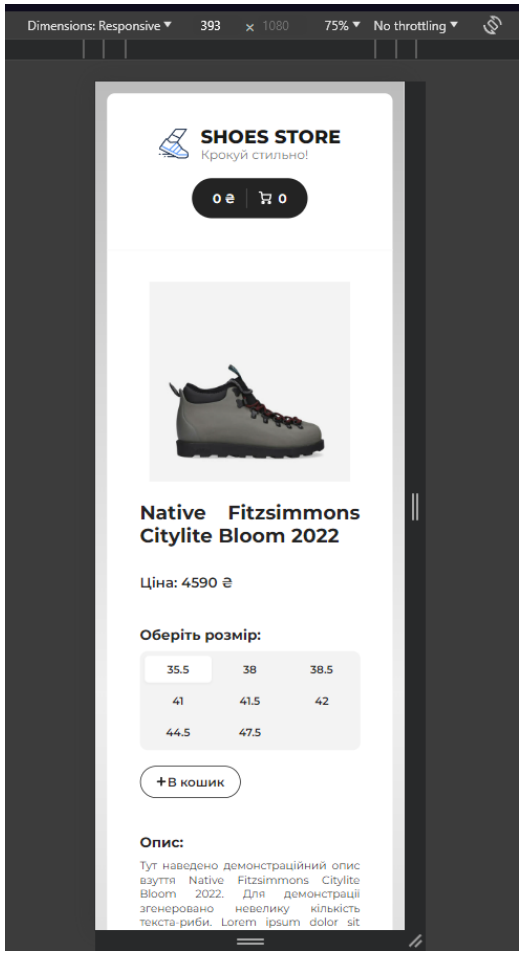


Рисунок 3.76 – Вигляд сторінки товару при ширині екрану 393 пікселів

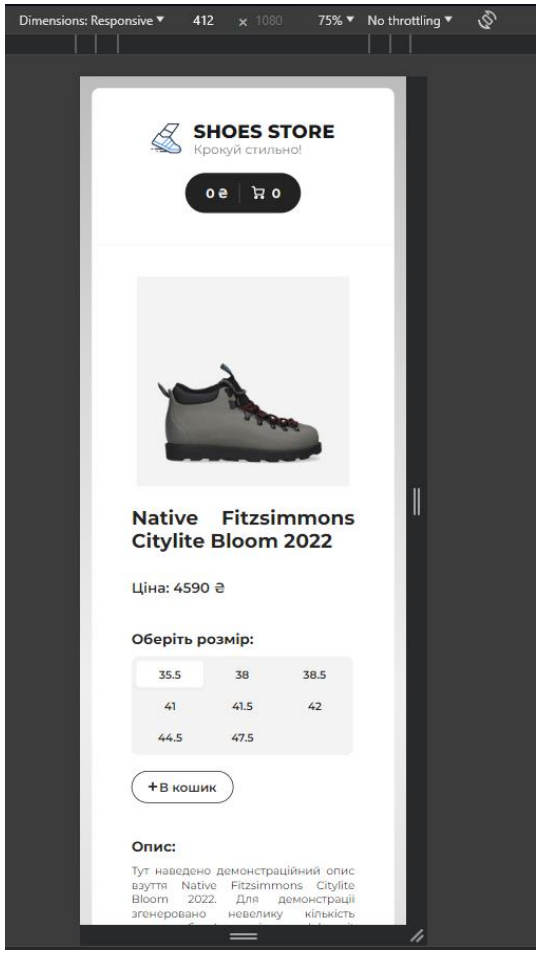


Рисунок 3.77 – Вигляд сторінки товару при ширині екрану 412 пікселів

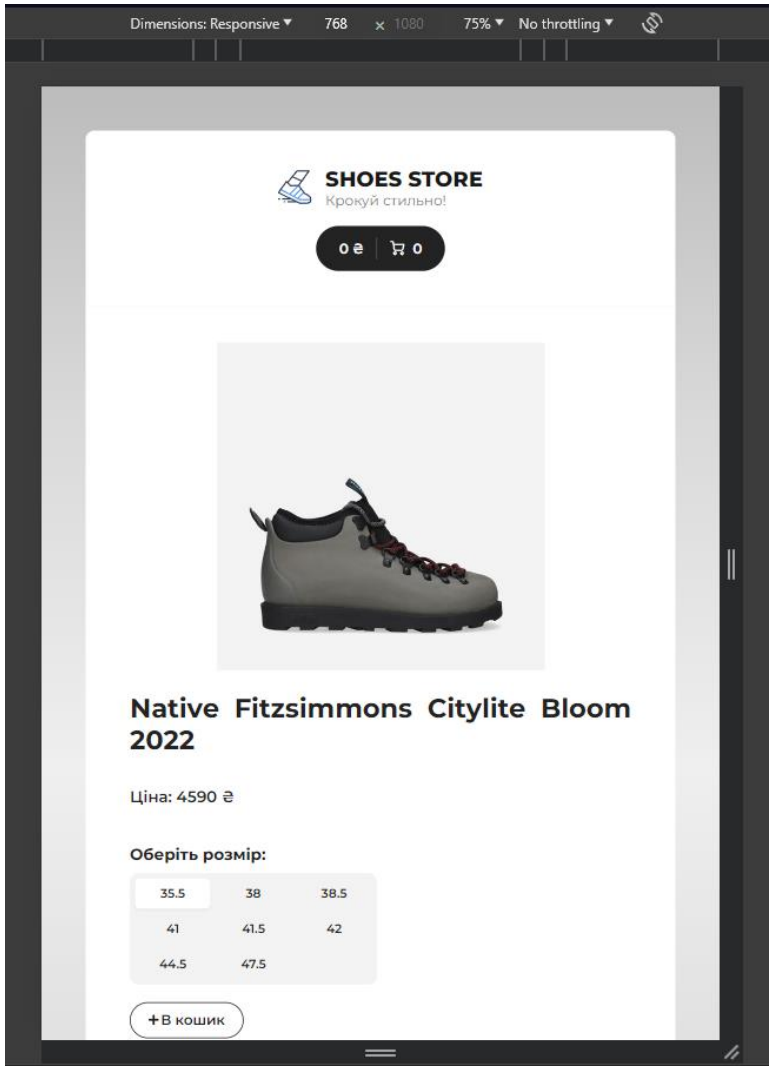


Рисунок **Error! No text of specified style in document.**3.78 – Вигляд сторінки товару при ширині екрану 768 пікселів

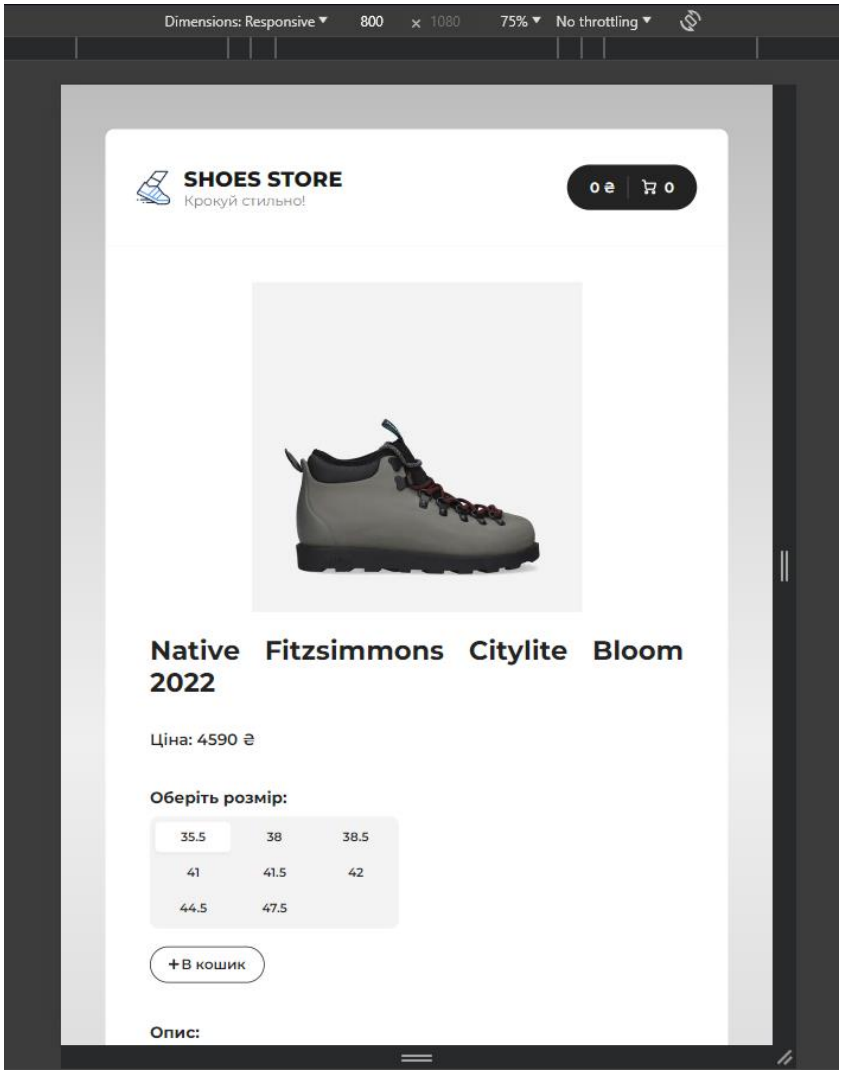


Рисунок 3.79 – Вигляд сторінки товару при ширині екрану 800 пікселів

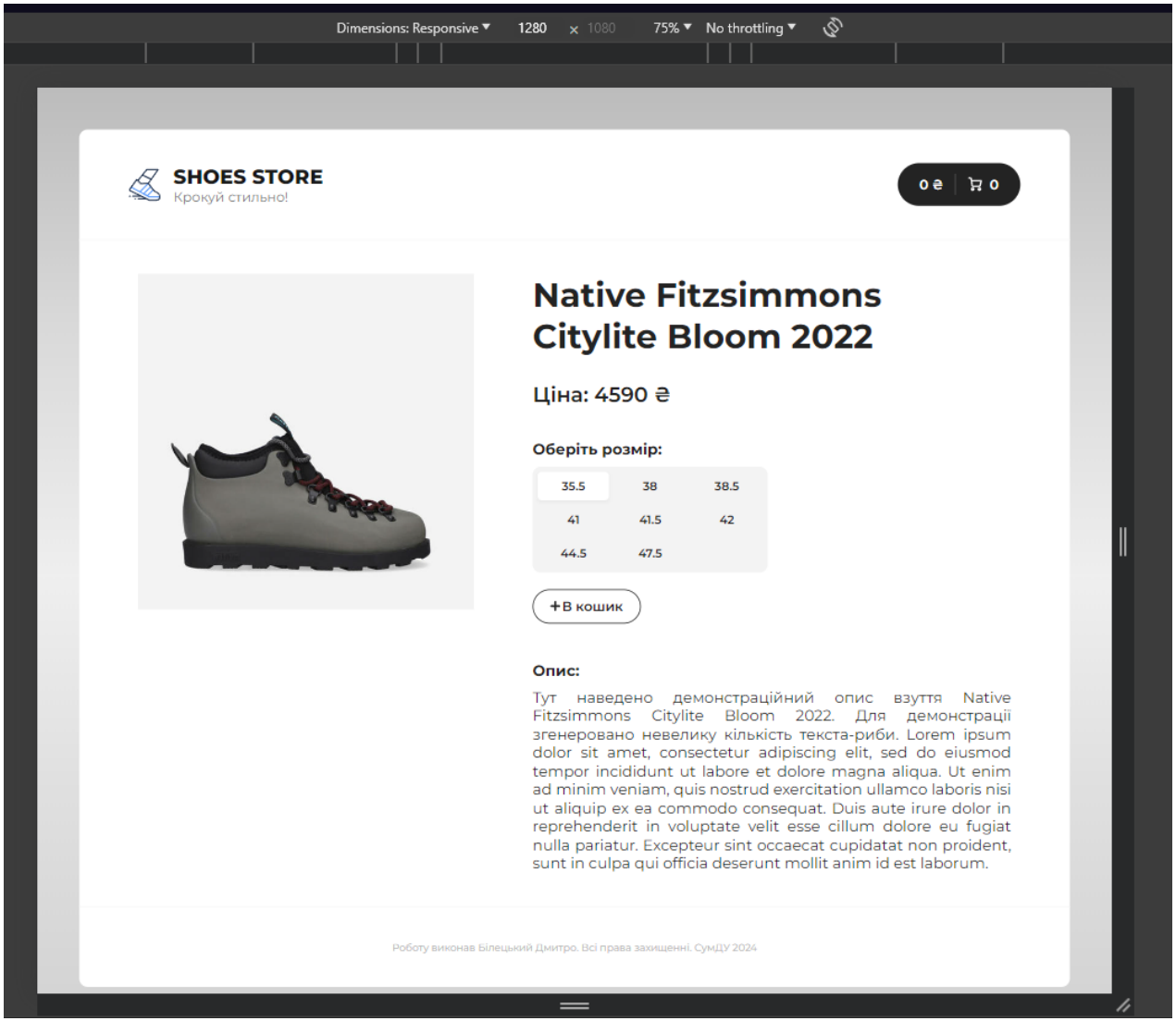


Рисунок **Error! No text of specified style in document.**3.80 – Вигляд сторінки товару при ширині екрану 1280 пікселів

Повний код стилізації наведено в git-репозитарії [19].

3.5.3. Сторінка кошику

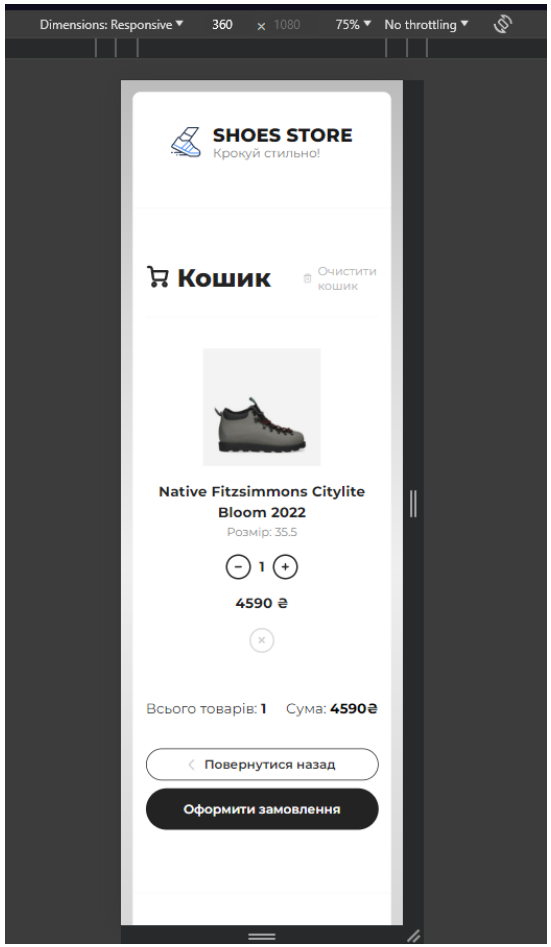


Рисунок 3.81 – Вигляд сторінки кошику при ширині екрану 360 пікселів

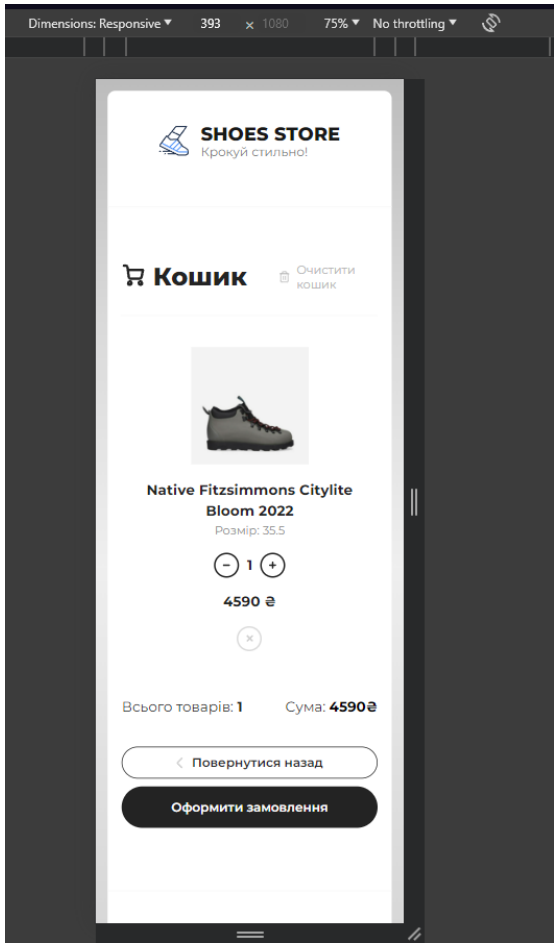


Рисунок 3.82 – Вигляд сторінки кошику при ширині екрану 393 пікселів

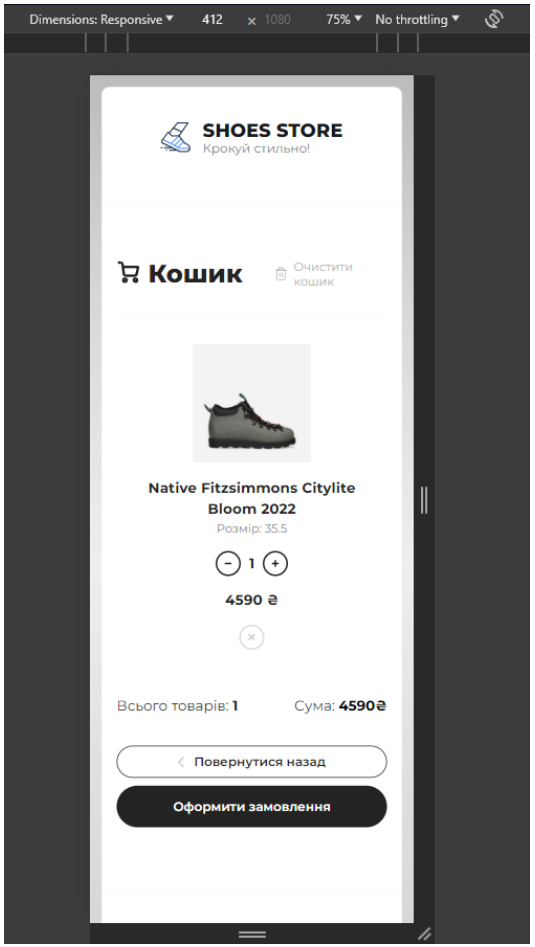


Рисунок 3.83 – Вигляд сторінки кошику при ширині екрану 412 пікселів

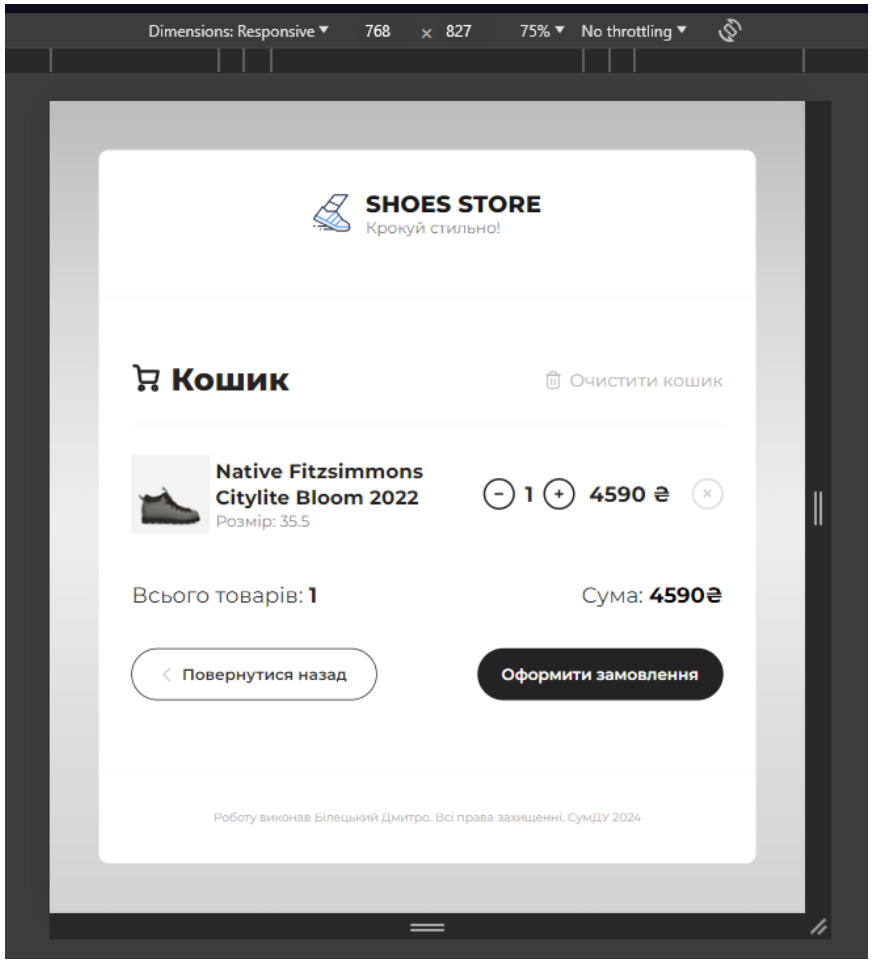


Рисунок 3.84 – Вигляд сторінки кошику при ширині екрану 768 пікселів

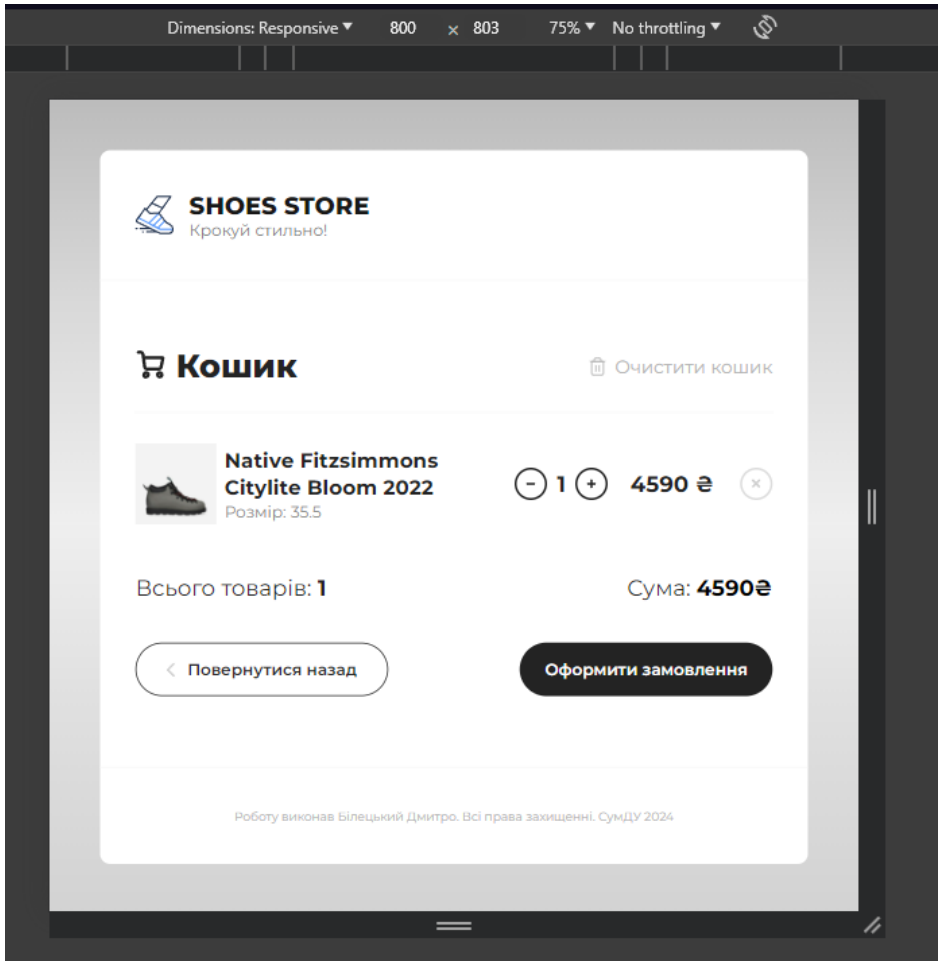


Рисунок 3.85 – Вигляд сторінки кошику при ширині екрану 800 пікселів

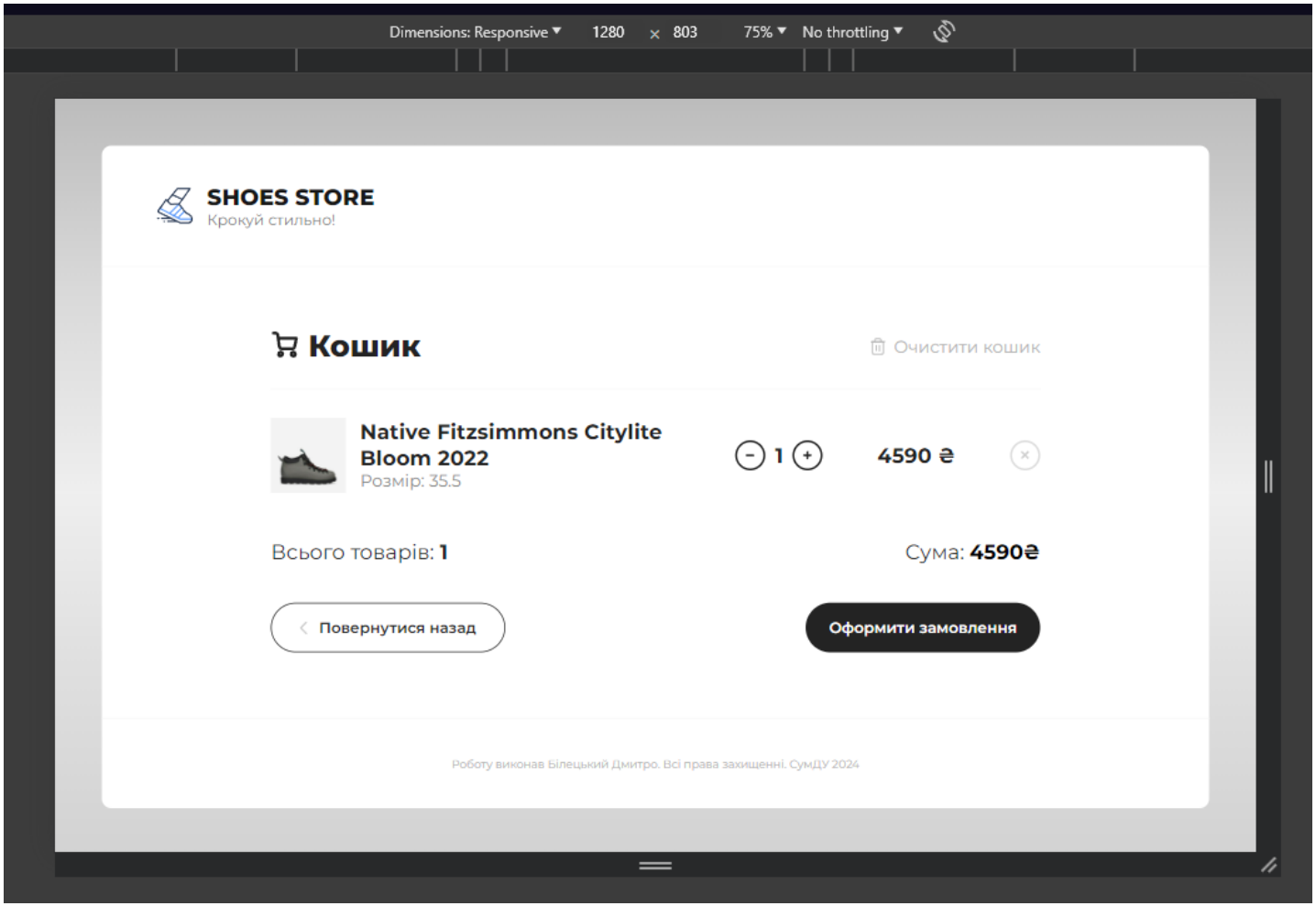


Рисунок **Error! No text of specified style in document.**3.86 – Вигляд сторінки кошику при ширині екрану 1280 пікселів

Повний код стилізації наведено в git-репозитарії [19].

3.6. Розгортування веб-додатку на хостингу

Для розгортування додатку було використано хостинг https://vercel.com/ [22]. Після авторизації на сайті використовуючи github аккаунт, наступним кроком необхідно імпортувати репозитарій проекту (рис. 3.87):

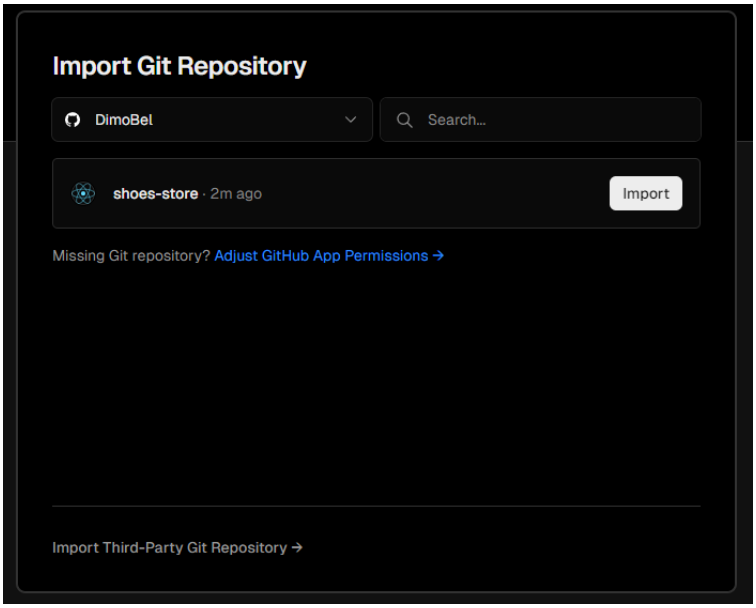


Рисунок **Error! No text of specified style in document.**3.87 – Імпортування git-репозитарію до Vercel

На сторінці, що відкрилася необхідно натиснути кнопку Deploy (рис. 3.88). Жодних налаштувань проект не потребує.

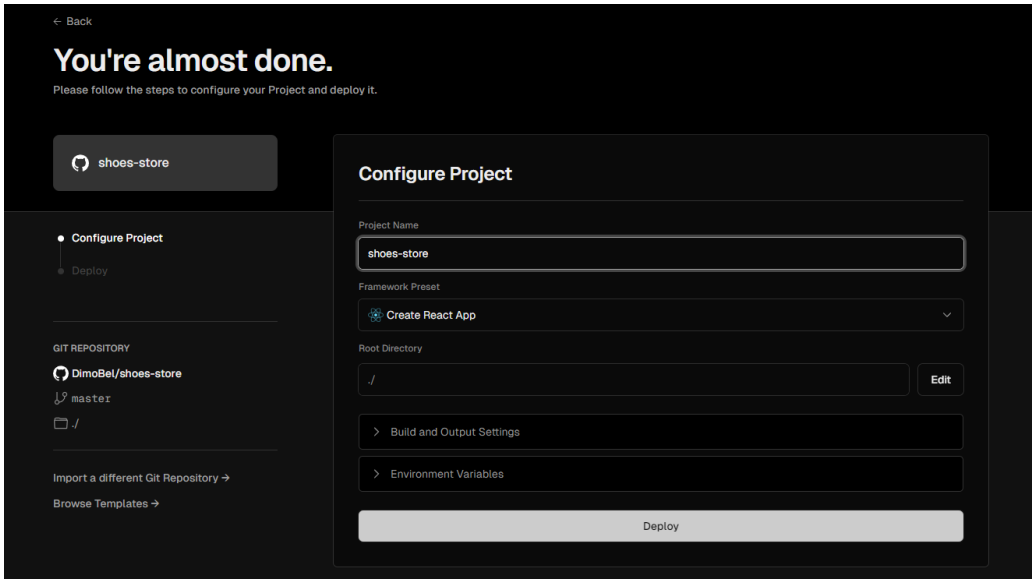


Рисунок **Error! No text of specified style in document.**3.88 – Розгортування проекту

Після цього, проект розгорнуто на хостингу Vercel (рис. 3.89):

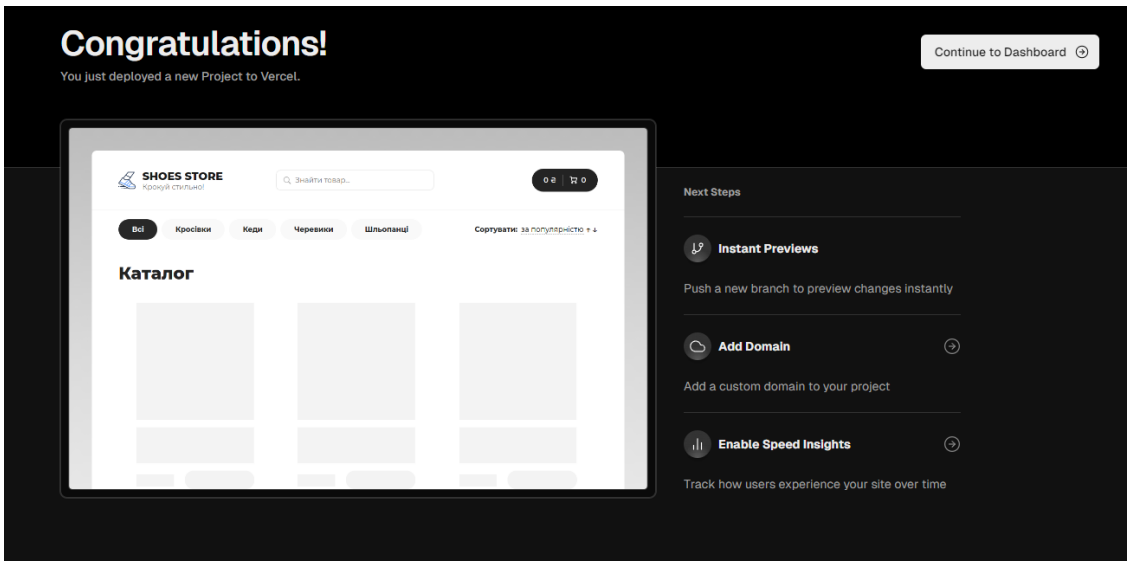


Рисунок **Error! No text of specified style in document.**3.89 – Результат успішного розгортання проекту

Веб-сайту призначено адресу https://shoes-store-omega.vercel.app [23], перейшовши за якою з ним можна взаємодіяти.

4. ВЗАЄМОДІЯ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Перейшовши за url-адресою https://shoes-store-omega.vercel.app [23], користувач потрапляє на домашню сторінку веб-сайту (рис. 4.1):

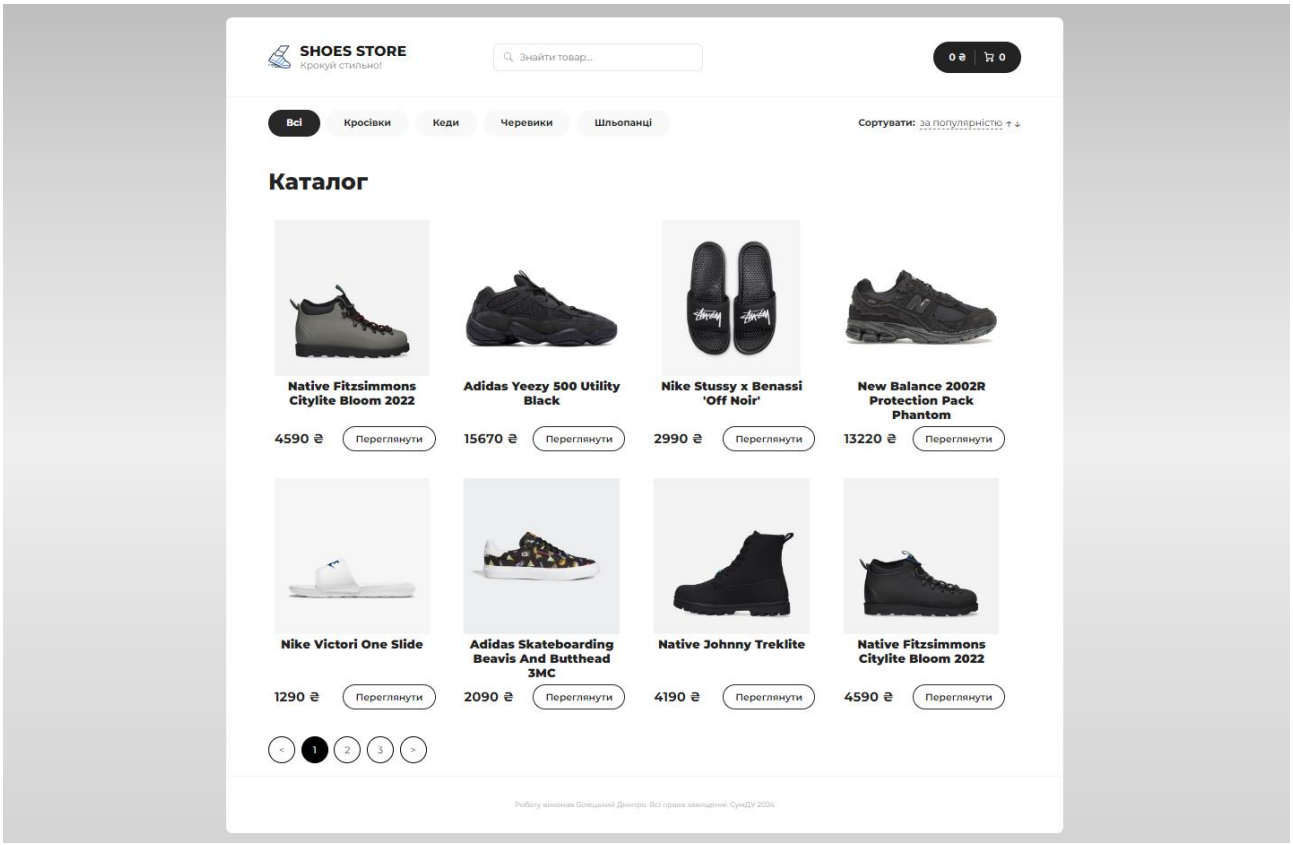


Рисунок **Error! No text of specified style in document.**4.1 – Вигляд головної сторінки веб-сайту

На домашній сторінці відображаються товари з усіх категорій, які за вмовчуванням відсортовані за популярністю за спаданням. Для того щоб, змінити параметри сортування, необхідно натиснути на поточний вид сортування, що розташований вище каталогу товарів. Відкриється випадаюче вікно де можна обрати потрібний вид сортування (рис. 4.2):

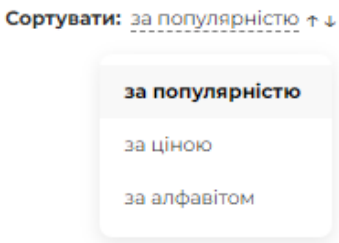
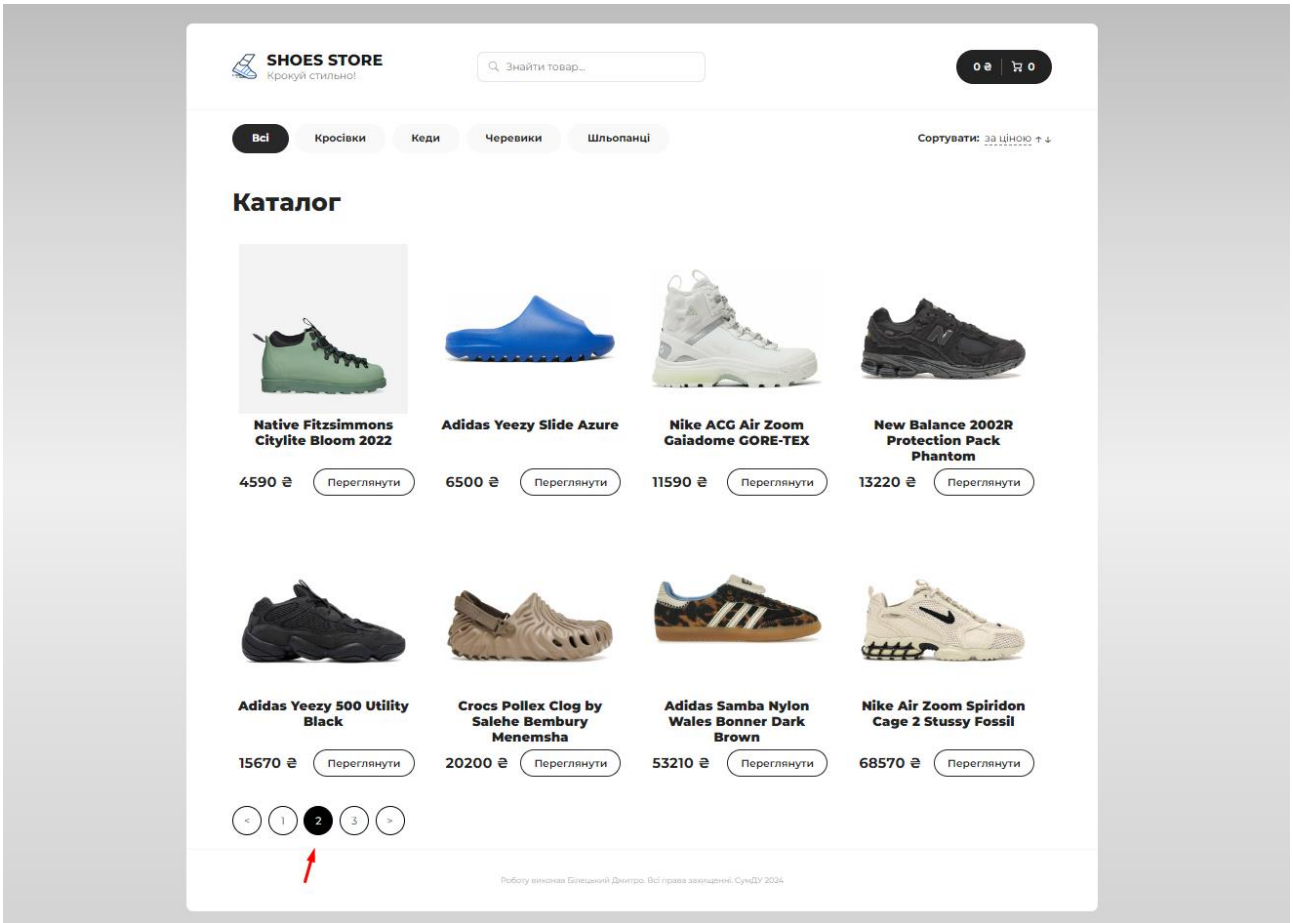


Рисунок **Error! No text of specified style in document.**4.2 – Види сортування товарів

Для того, щоб обрати тип сортування товарів (за зростанням чи за спаданням) необхідно натиснути відповідні кнопки у вигляді стрілок, що розташовані правіше поточного типу сортування.

Для того, щоб перейти на іншу сторінку каталогу, необхідно скористатися панеллю пагінації, що розташована нижче каталогу товарів. Натиснувши на номер необхідної сторінки користувач побачить товари, які на ній знаходяться (рис. 4.3):



Рисунок**Error! No text of specified style in document.** 4.3 – Товари з усіх категорій на другій сторінці каталогу

Для зміни категорії товарів необхідно скористатися навігаційною панеллю, що знаходиться вище каталогу. Для переходу на категорію, що цікавить, необхідно натиснути кнопку з відповідною назвою категорії (рис. 4.4):

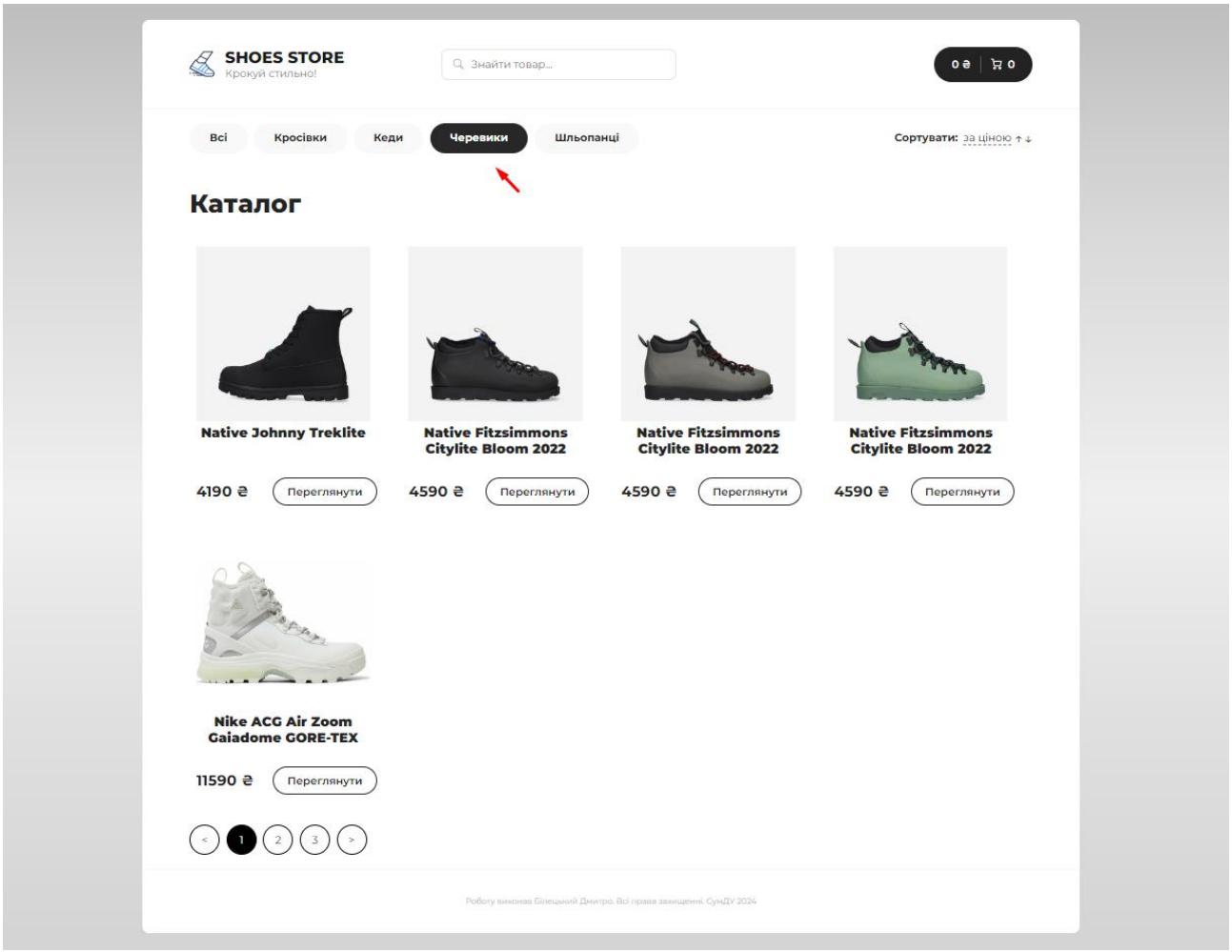


Рисунок **Error! No text of specified style in document.**4.4 – Товари категорії Черевики

Крім того, як можна побачити на рисунку 4.4, сортувати товари можна і в межах окремої категорії. В даному випадку вони відсортовані за ціною за зростанням.

Для пошуку товарів, що цікавлять, необхідно скористатися полем пошуку, що розташовано в центральній частині хедеру (рис. 4.5):

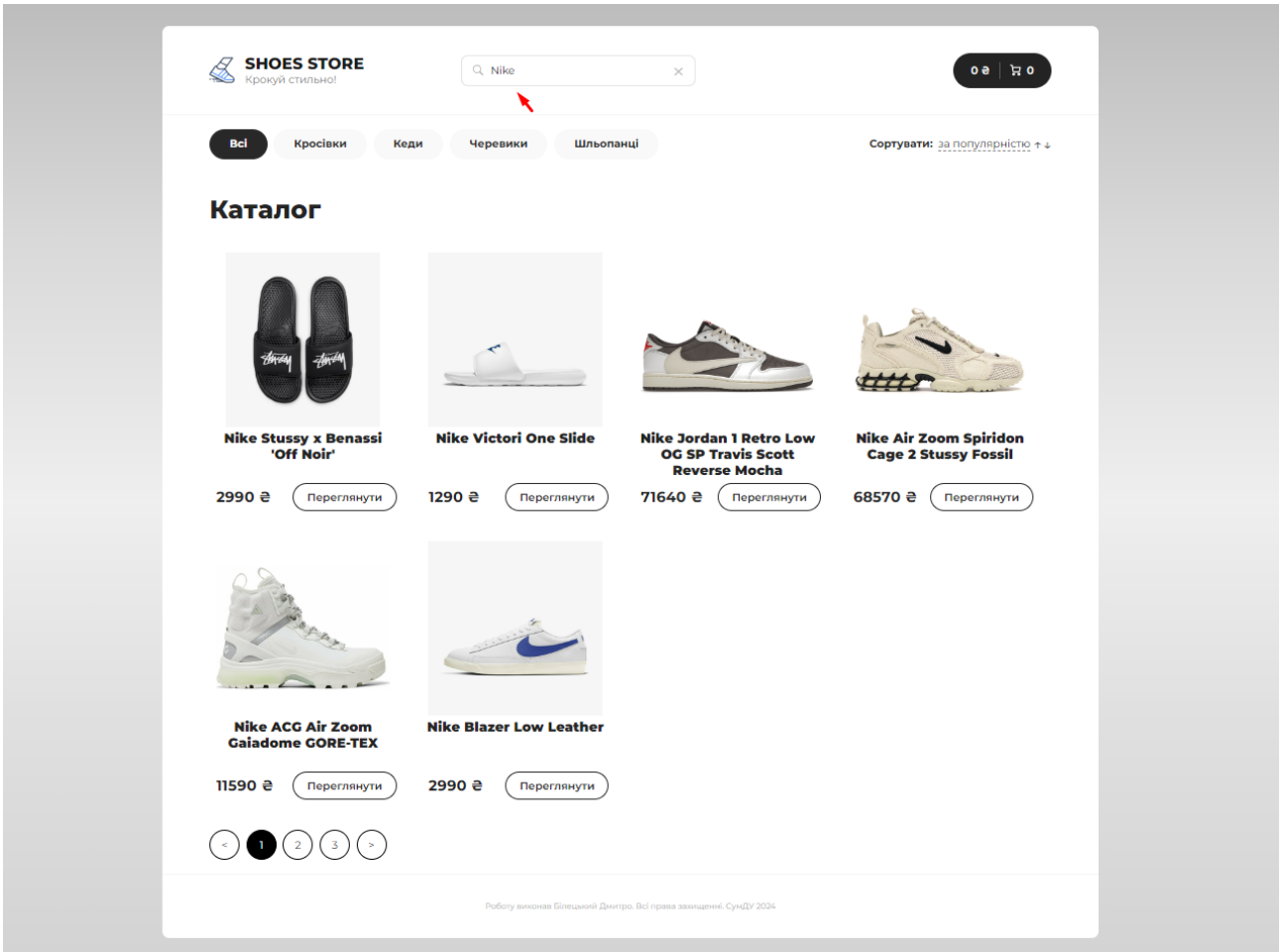


Рисунок **Error! No text of specified style in document.**4.5 – Пошук потрібних товарів

У випадку, якщо користувач введе некоректний пошуковий запит, або потрібного товару просто не буде в інтернет-магазині, він побачить відповідне інформаційне повідомлення (рис. 4.6):

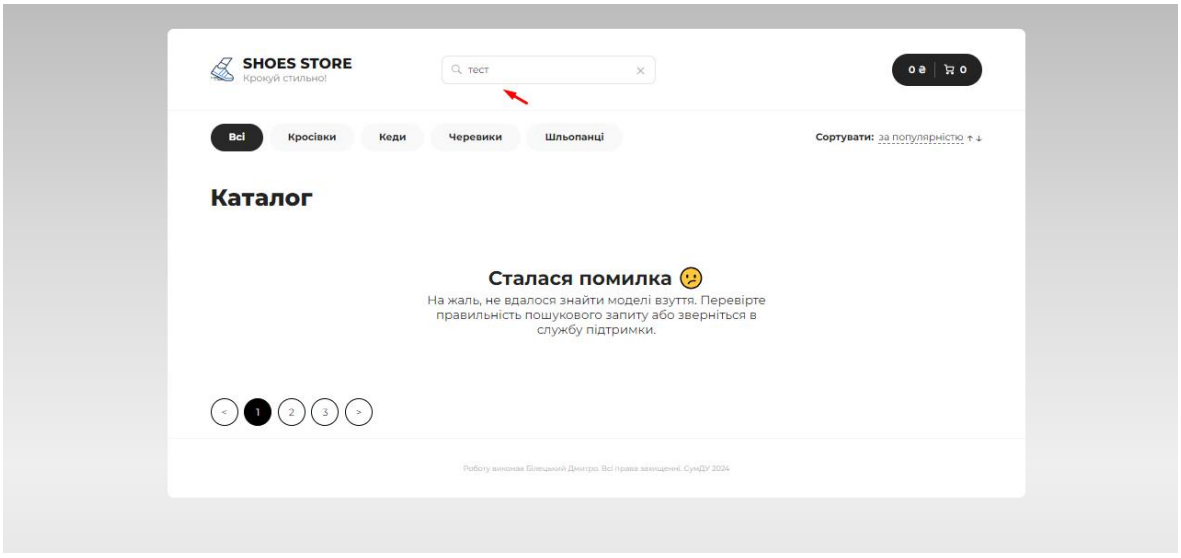


Рисунок **Error! No text of specified style in document.**4.6 – Інформаційне повідомлення про неможливість пошуку товарів

Натиснувши на кнопку «Переглянути» або на фотографію товару, що цікавить, в каталозі товарів користувач потрапить на сторінку товару (рис. 4.7):

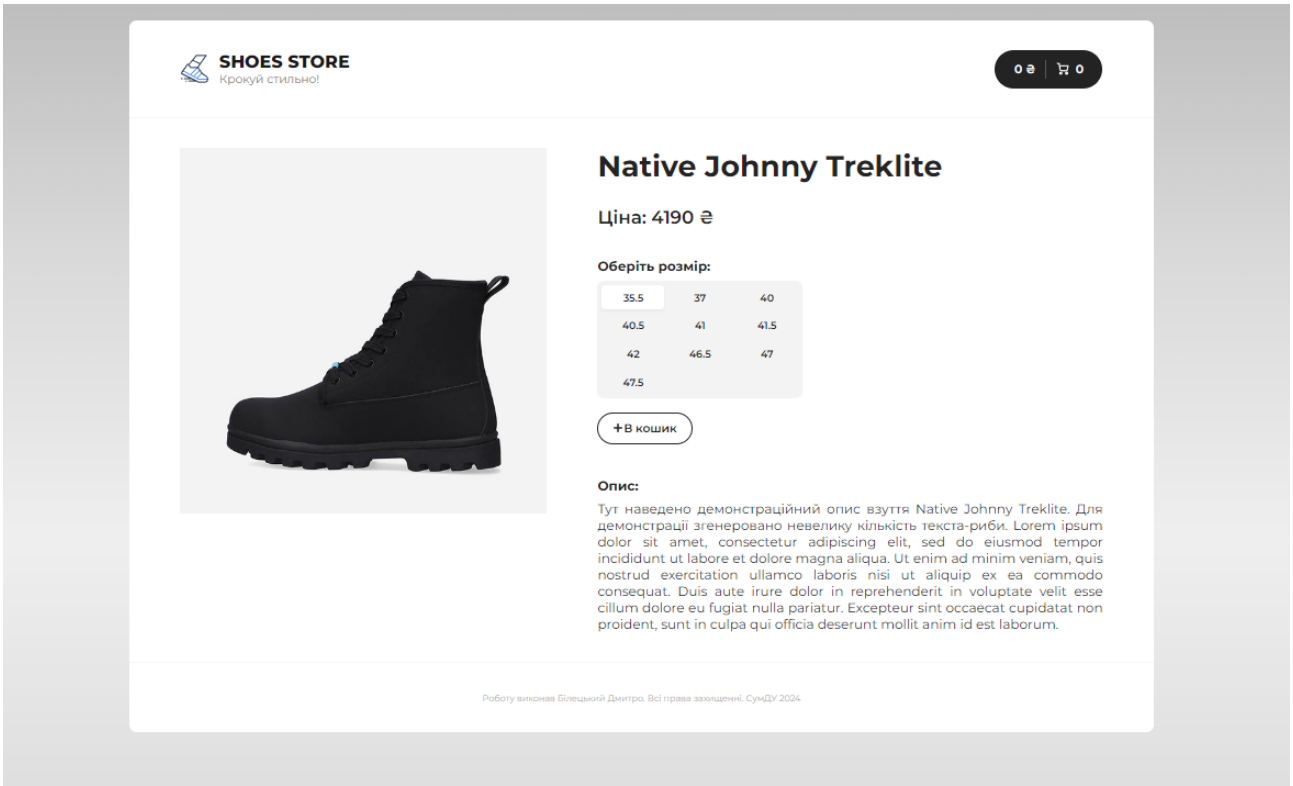


Рисунок **Error! No text of specified style in document.**4.7 – Вигляд сторінки товару

На сторінці товару користувач може ознайомитися з детальним описом товару, обрати розмір серед доступних у таблиці розмірів, і додати товар до кошику.

Для того, щоб перейти до сторінки кошику товарів необхідно натиснути кнопку, що розташована в правій частині хедеру. Сторінка кошику товарів має наступний вигляд (рис. 4.8):

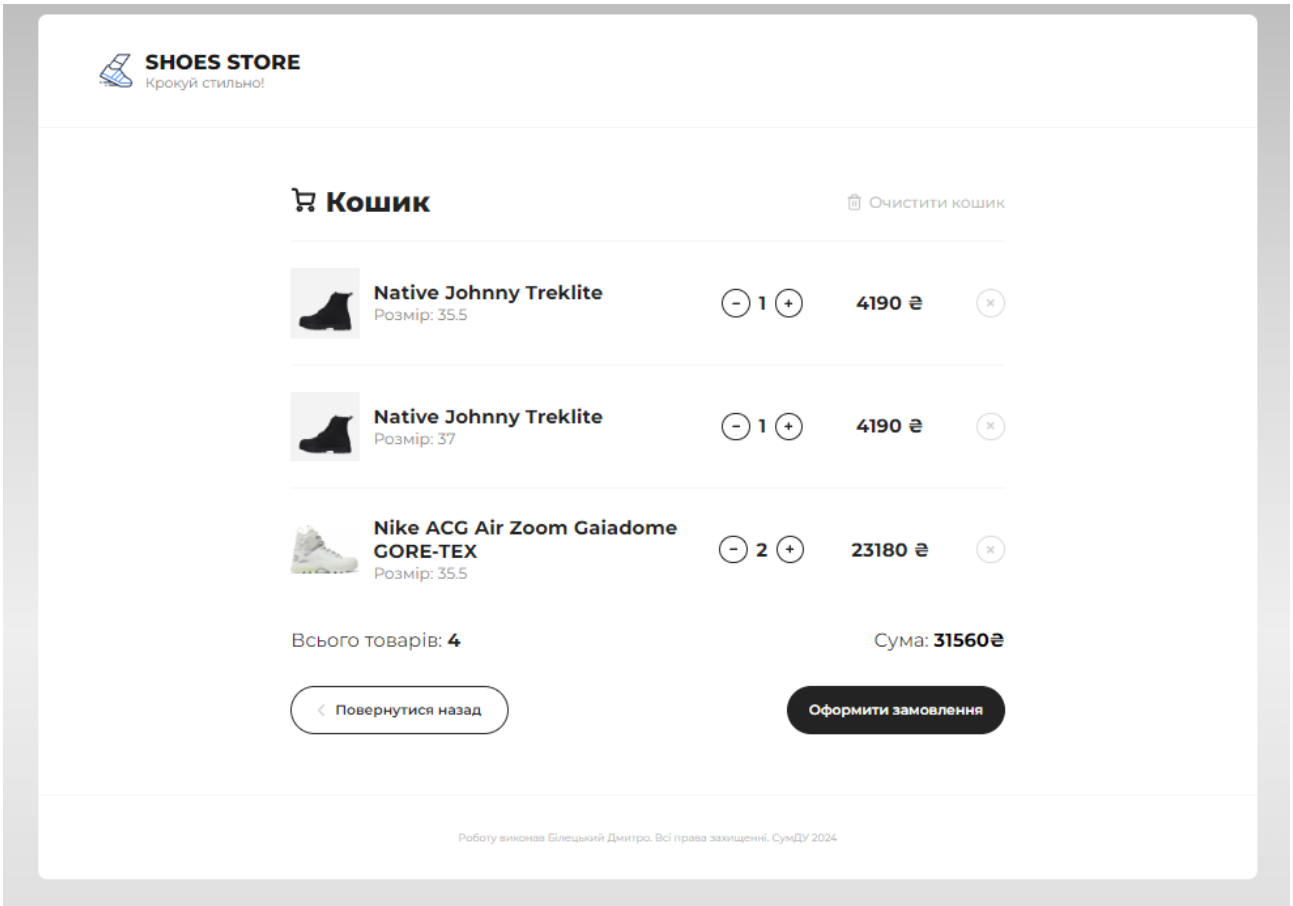


Рисунок **Error! No text of specified style in document.**4.8 – Вигляд сторінки кошику товарів

На сторінці кошику товарів користувач може побачити додані раніше товари, враховуючи окремі розміри конкретних моделей, їх загальну кількість і вартість. Крім того, користувач може змінити кількість конкретних товарів, використовуючи кнопки «-» і «+», що розташовані біля кількості товару, може видалити товар з кошика використовуючи кнопку «×», що розташована справа від ціни товару, та очистити весь кошик використовуючи кнопку «Очистити кошик».

У випадку, якщо користувач спробує перейти на сторінку, якої не існує в межах веб-сайту, він побачить відповідне повідомлення (рис. 4.9):

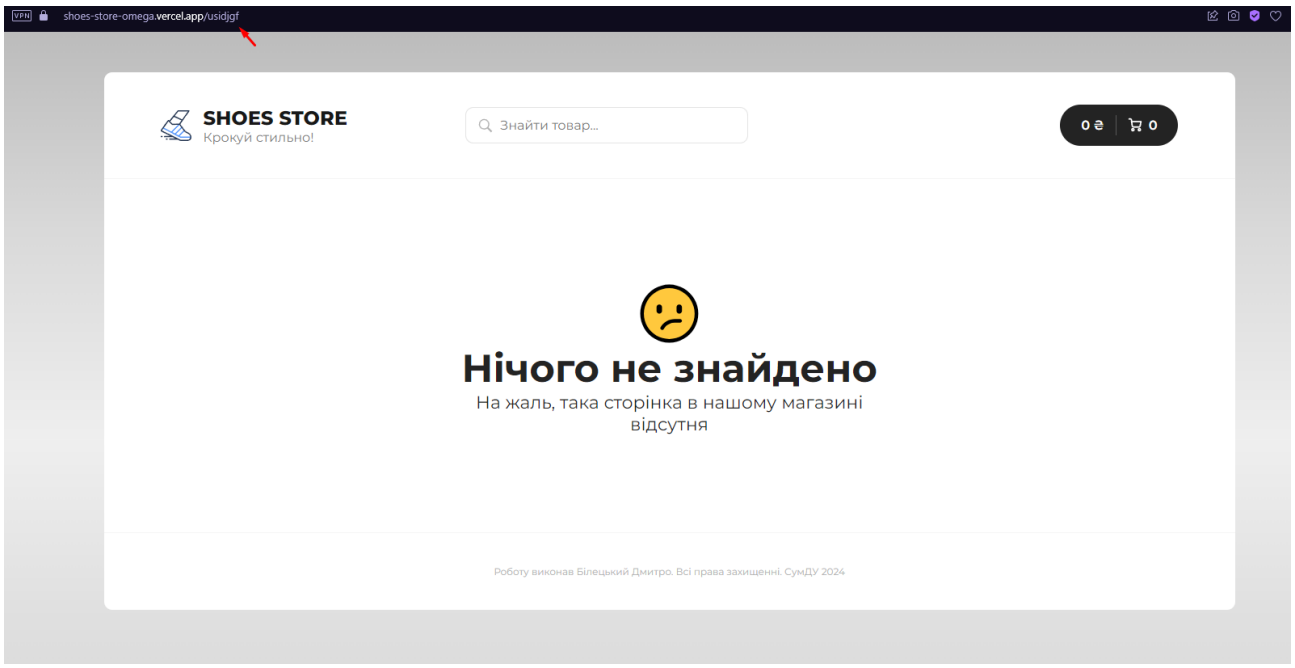


Рисунок **Error! No text of specified style in document.**4.9 – Вигляд сторінки 404

Завдяки реалізації сторінки 404 користувач залишиться в межах інтернет-магазину та зможе повернутися на головну сторінку, натиснувши на логотип або назву сайту, що розташовані в правій частині хедеру.

ВИСНОВКИ

Основною метою роботи є виконання завдання та досягнення цілей проекту, а саме створення адаптивного веб-сайту для продажу взуття, з реалізацією функціоналу пошуку товарів, розбиття їх на категорії, сортування товарів за різними параметрами, а також забезпечення сторінки кожного товару і функціоналу кошику товарів, забезпечення адаптивної верстки всього веб-додатку, розгорнення веб-сайту на хостингу.

Під час виконання даної роботи було проведено аналіз предметної області, а саме інтернет-магазинів зі схожим призначенням, та визначено актуальність роботи.

Було створено модель інформаційної системи, а саме визначено структуру сторінок веб-сайту та їх компонентів, які було реалізовано.

Реалізовано розбиття товарі на категорії, їх пошук, сортування за різними параметрами.

Забезпечено відображення окремої сторінки з детальним описом кожного товару.

Забезпечено адаптивну верстку для комфортного користування веб-сайтом на будь-якій роздільній здатності екрана та розгорнуто веб-додаток на хостингу для можливості безпосередньої взаємодії з ним.

Для програмної реалізації були використані сучасні бібліотеки та інструменти для розробки веб-сайтів, що забезпечило кращий користувацький досвід.

Цей проект створює основу для подальшого розширення функціоналу, такого як створення власного бекенду, що забезпечить користувачам можливість оформлення замовлення.

Робота виконана в повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Майже 23 млн українців регулярно користуються Інтернетом – дослідження. Mind.ua. URL: https://mind.ua/news/20204323-majzhe-23-mln-ukrayinciv-regulyarno-koristuyutsya-internetom-doslidzhennya (дата звернення: 31.04.2024).
2. LOVE&LIVE. Одяг від українських дизайнерів і брендів: купити в інтернет-магазині LOVE&LIVE. URL: https://loveandlive.ua/ (дата звернення: 15.04.2024).
3. Інтернет-магазин одягу в Україні: роздріб, опт - Assorti. Інтернет-магазин одягу в Україні: роздріб, опт - Assorti. URL: https://assorti-odessa.com.ua/ua/ (дата звернення: 15.04.2024).
4. Cropp Україна. CROPP official online store | Choose your country. URL: https://www.cropp.com/ua/uk/ (дата звернення: 15.04.2024).
5. Баліцька Г. І. Кваліфікаційна робота на здобуття ступеня магістра. Особливості розвитку інтернет-торгівлі в Україні. URL: https://econom.lnu.edu.ua/wp-content/uploads/2022/10/Balitska-H.I.-EknM-21s.pdf (дата звернення: 01.05.2024).
6. Що таке React JS і для чого він потрібен?. dan-it.com.ua. URL: https://dan-it.com.ua/uk/blog/chto-takoe-react-js-i-dlja-chego-on-nuzhen/ (дата звернення: 08.04.2024).
7. Feature Overview v6.23.0. React Router. URL: https://reactrouter.com/en/main/start/overview (дата звернення: 08.04.2024).
8. Redux Fundamentals, Part 1: Redux Overview | Redux. Redux - A JS library for predictable and maintainable global state management | Redux. URL: https://redux.js.org/tutorials/fundamentals/part-1-overview (дата звернення: 09.04.2024).
9. Redux Fundamentals, Part 3: State, Actions, and Reducers | Redux. Redux - A JS library for predictable and maintainable global state management | Redux. URL:

https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers (дата звернення: 09.04.2024).
10. Query String | Node.js v6.17.1 Documentation. Node.js – Run JavaScript Everywhere. URL: https://nodejs.org/dist/latest-v6.x/docs/api/querystring.html (дата звернення: 10.04.2024).
11. Sass Introduction. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/sass/sass_intro.php (дата звернення: 10.04.2024).
12. mockAPI. mockAPI. URL: https://mockapi.io (дата звернення: 10.04.2024).
13. Початок | Axios Docs. Axios. URL: https://axios-http.com/uk/docs/intro (дата звернення: 09.04.2024).
14. Vercel Documentation. Vercel Documentation. URL: https://vercel.com/docs (дата звернення: 10.04.2024).
15. Create React App. Create React App. URL: https://create-react-app.dev (дата звернення: 11.04.2024).
16. Usage Guide | Redux Toolkit. Redux Toolkit | Redux Toolkit. URL: https://redux-toolkit.js.org/usage/usage-guide#store-setup (дата звернення: 09.04.2024).
17. State: A Component's Memory – React. React. URL: https://react.dev/learn/state-a-components-memory (дата звернення: 01.05.2024).
18. React Skeleton - Create Content Loader. React Skeleton - Create Content Loader. URL: https://skeletonreact.com (дата звернення: 15.04.2024).
19. DimoBel/shoes-store. GitHub. URL: https://github.com/DimoBel/shoes-store
20. Mobile Screen Resolution Stats Ukraine | Statcounter Global Stats. StatCounter Global Stats. URL: https://gs.statcounter.com/screen-resolution-stats/mobile/ukraine (дата звернення: 06.05.2024).
21. Tablet Screen Resolution Stats Ukraine | Statcounter Global Stats. StatCounter Global Stats. URL: https://gs.statcounter.com/screen-resolution-stats/tablet/ukraine (дата звернення: 06.05.2024).
22. Vercel: Build and deploy the best Web experiences with The Frontend Cloud. Vercel. URL: https://vercel.com (дата звернення: 07.05.2024).

23. Shoes Store. Shoes Store. URL: https://shoes-store-omega.vercel.app/

ДОДАТКИ

Повний код веб-сайту наведено в git-репозитарії [19].

Додаток А. shoes.json


```
[
 {
  "id": "0",
  "imageUrl": "https://sneakersbanda.ua/images/thumbnails/269/269/detailed/343/New-Balance-M2002-Protection-Pack-Phantom-Product_j8bf-kw.jpg",
  "title": "New Balance 2002R Protection Pack Phantom",
  "sizes": [43, 43.5, 44, 44.5, 45, 45.5],
  "price": 13220,
  "category": 1,
  "rating": 8,
  "description": "Тут наведено демонстраційний опис взуття New Balance 2002R Protection Pack Phantom. Для демонстрації згенеровано невелику кількість текста-риби. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
 },
 {
  "id": "1",
  "imageUrl": "https://sneakersbanda.ua/images/thumbnails/530/530/detailed/340/Air-Jordan-1-Retro-Low-OG-SP-Travis-Scott-Reverse-Mocha_V2-Product_nvk1-uk.jpg",
  "title": "Nike Jordan 1 Retro Low OG SP Travis Scott Reverse Mocha",
  "sizes": [38.5, 39, 39.5, 40, 40.5, 41, 41.5, 42, 42.5, 43],
  "price": 71640,
  "category": 1,
  "rating": 3,
  "description": "Тут наведено демонстраційний опис взуття Nike Jordan 1 Retro Low OG SP Travis Scott Reverse Mocha. Для демонстрації згенеровано невелику кількість текста-риби. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
 },
 {
  "id": "2",
  "imageUrl": "https://sneakersbanda.ua/images/thumbnails/530/530/detailed/1850/adidas-Samba-Nylon-Wales-Bonner-Dark-Brown-Product.jpg",
  "title": "Adidas Samba Nylon Wales Bonner Dark Brown",
  "sizes": [38.5, 40, 40.5, 41, 43],
  "price": 53210,
  "category": 1,
  "rating": 5,
  "description": "Тут наведено демонстраційний опис взуття Adidas Samba Nylon Wales Bonner Dark Brown. Для демонстрації згенеровано невелику кількість текста-риби. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
 },
 {
  "id": "3",
  "imageUrl": "https://sneakersbanda.ua/images/thumbnails/530/530/detailed/1598/Nike-Air-Zoom-Spiridon-Cage-2-Stussy-Fossil-Product.jpg",
  "title": "Nike Air Zoom Spiridon Cage 2 Stussy Fossil",
  "sizes": [37.5, 38, 39.5, 41, 43, 43.5, 47, 47.5],
  "price": 68570,
  "category": 1,
  "rating": 2,
  "description": "Тут наведено демонстраційний опис взуття Nike Air Zoom Spiridon Cage 2 Stussy Fossil. Для демонстрації згенеровано невелику кількість текста-риби. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore
```

```
magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
  },
  {
    "id": "4",
    "imageUrl": "https://sneakersbanda.ua/images/thumbnails/530/530/detailed/367/Adidas-Yeezy-500-Utility-Black-Product_wapj-t0.jpg",
    "title": "Adidas Yeezy 500 Utility Black",
    "sizes": [42, 42.5, 43],
    "price": 15670,
    "category": 1,
    "rating": 9,
    "description": "Тут наведено демонстраційний опис взуття Adidas Yeezy 500 Utility Black. Для демонстрації згенеровано невелику кількість текста-риби. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
  },
  {
    "id": "5",
    "imageUrl": "https://sneakersbanda.ua/images/thumbnails/530/530/detailed/27/CI6377_107_a.jpg",
    "title": "Nike Blazer Low Leather",
    "sizes": [43, 43.5, 44],
    "price": 2990,
    "category": 2,
    "rating": 1,
    "description": "Тут наведено демонстраційний опис взуття Nike Blazer Low Leather. Для демонстрації згенеровано невелику кількість текста-риби. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
  },
  {
    "id": "6",
    "imageUrl":
"https://sneakersbanda.ua/images/thumbnails/530/530/detailed/151/3MC_x_Beavis_and_Butthead_Shoes_Black_BD7861_01_standard.jpg",
    "title": "Adidas Skateboarding Beavis And Butthead 3MC",
    "sizes": [35.5, 36, 36.5, 37, 37.5, 46, 46.5, 47, 47.5],
    "price": 2090,
    "category": 2,
    "rating": 7,
    "description": "Тут наведено демонстраційний опис взуття Adidas Skateboarding Beavis And Butthead 3MC. Для демонстрації згенеровано невелику кількість текста-риби. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
  },
  {
    "id": "7",
    "imageUrl": "https://sneakersbanda.ua/images/thumbnails/530/530/detailed/152/61Qw4Vy-UXL._UL1500_.jpg",
    "title": "Puma Careofxpuma Court L",
    "sizes": [37, 37.5],
    "price": 2590,
    "category": 2,
    "rating": 4,
    "description": "Тут наведено демонстраційний опис взуття Puma Careofxpuma Court L. Для демонстрації згенеровано невелику кількість текста-риби. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
  },
  {
    "id": "8",
```

```
  "imageUrl": "https://sneakersbanda.ua/images/thumbnails/530/530/detailed/571/ukr_pl_VZUTTIA-Native-FITZSIMMONS-CITYLITE-BLOOM-31106848-1019-1050457_1.jpg",
  "title": "Native Fitzsimmons Citylite Bloom 2022",
  "sizes": [35.5, 36, 39.5, 40, 40.5, 41, 41.5, 47.5],
  "price": 4590,
  "category": 3,
  "rating": 6,
  "description": "Тут наведено демонстраційний опис взуття Native Fitzsimmons Citylite Bloom 2022. Для демонстрації згенеровано невелику кількість текста-риби. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
 },
 {
  "id": "9",
  "imageUrl": "https://sneakersbanda.ua/images/thumbnails/530/530/detailed/571/ukr_pm_CHEREVIKI-Native-Fitzsimmons-Citylite-Bloom-31106848-1233-1050530_1.jpg",
  "title": "Native Fitzsimmons Citylite Bloom 2022",
  "sizes": [35.5, 38, 38.5, 41, 41.5, 42, 44.5, 47.5],
  "price": 4590,
  "category": 3,
  "rating": 10,
  "description": "Тут наведено демонстраційний опис взуття Native Fitzsimmons Citylite Bloom 2022. Для демонстрації згенеровано невелику кількість текста-риби. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
 },
 {
  "id": "10",
  "imageUrl": "https://sneakersbanda.ua/images/thumbnails/530/530/detailed/1838/1066866_01.jpg.jpeg",
  "title": "Nike ACG Air Zoom Gaiadome GORE-TEX",
  "sizes": [35.5, 36, 38, 38.5, 41.5, 42, 45, 45.5, 47.5],
  "price": 11590,
  "category": 3,
  "rating": 2,
  "description": "Тут наведено демонстраційний опис взуття Nike ACG Air Zoom Gaiadome GORE-TEX. Для демонстрації згенеровано невелику кількість текста-риби. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
 },
 {
  "id": "11",
  "imageUrl": "https://sneakersbanda.ua/images/thumbnails/530/530/detailed/207/eng_pl_Native-Johnny-Treklite-41108330-1001-1030529_3.jpg",
  "title": "Native Johnny Treklite",
  "sizes": [35.5, 37, 40, 40.5, 41, 41.5, 42, 46.5, 47, 47.5],
  "price": 4190,
  "category": 3,
  "rating": 7,
  "description": "Тут наведено демонстраційний опис взуття Native Johnny Treklite. Для демонстрації згенеровано невелику кількість текста-риби. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
 },
 {
  "id": "12",
  "imageUrl": "https://sneakersbanda.ua/images/thumbnails/530/530/detailed/571/ukr_pl_CHEREVIKI-Native-Fitzsimmons-Citylite-Bloom-31106848-3323-1051381_2.jpg",
  "title": "Native Fitzsimmons Citylite Bloom 2022",
  "sizes": [35.5, 36, 39.5, 40, 40.5],
  "price": 4590,
```

```
  "category": 3,
  "rating": 4,
  "description": "Тут наведено демонстраційний опис взуття Native Fitzsimmons Citylite Bloom 2022. Для демонстрації згенеровано невелику кількість текста-риби. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
 },
 {
  "id": "13",
  "imageUrl": "https://sneakersbanda.ua/images/thumbnails/530/530/detailed/1817/1042587_01.jpg.jpeg",
  "title": "Adidas Yeezy Slide Azure",
  "sizes": [38, 38.5, 39, 39.5],
  "price": 6500,
  "category": 4,
  "rating": 1,
  "description": "Тут наведено демонстраційний опис взуття Adidas Yeezy Slide Azure. Для демонстрації згенеровано невелику кількість текста-риби. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
 },
 {
  "id": "14",
  "imageUrl": "https://sneakersbanda.ua/images/thumbnails/530/530/detailed/373/Crocs-Pollex-Clog-by-Salehe-Bembury-Menemsha-Product.jpg",
  "title": "Crocs Pollex Clog by Salehe Bembury Menemsha",
  "sizes": [37.5, 38, 46, 46.5, 47, 47.5],
  "price": 20200,
  "category": 4,
  "rating": 5,
  "description": "Тут наведено демонстраційний опис взуття Crocs Pollex Clog by Salehe Bembury Menemsha. Для демонстрації згенеровано невелику кількість текста-риби. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
 },
 {
  "id": "15",
  "imageUrl": "https://sneakersbanda.ua/images/thumbnails/530/530/detailed/1817/benassi-x-stssy-off-noir-release-date.jpg",
  "title": "Nike Stussy x Benassi 'Off Noir'",
  "sizes": [36.5, 37, 37.5, 41.5, 42, 42.5],
  "price": 2990,
  "category": 4,
  "rating": 9,
  "description": "Тут наведено демонстраційний опис взуття Nike Stussy x Benassi 'Off Noir'. Для демонстрації згенеровано невелику кількість текста-риби. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
 },
 {
  "id": "16",
  "imageUrl": "https://sneakersbanda.ua/images/thumbnails/530/530/detailed/1916/CN9675_102_a.jpg",
  "title": "Nike Victori One Slide",
  "sizes": [38, 38.5, 45.5, 46, 46.5, 47, 47.5],
  "price": 1290,
  "category": 4,
  "rating": 8,
  "description": "Тут наведено демонстраційний опис взуття Nike Victori One Slide. Для демонстрації згенеровано невелику кількість текста-риби. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."
```

```
 }
]
```

Додаток Б. shoesSlice.js

```
import { createAsyncThunk, createSlice } from '@reduxjs/toolkit';
import axios from 'axios';

export const fetchShoes = createAsyncThunk('shoes/fetchShoesStatus', async (params, thunkAPI) => {
  const { sortBy, order, category, search, currentPage } = params;
  const { data } = await axios.get(
    `https://66264d33052332d55322633f.mockapi.io/items?page=${currentPage}&limit=8&${category}&sortBy=${sortBy}&order=${order}&${search
}`,
  );
  return data;
});

const initialState = {
  items: [],
  status: 'loading',
};

const shoesSlice = createSlice({
  name: 'shoes',
  initialState,
  reducers: {
    setItems(state, action) {
      state.items = action.payload;
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(fetchShoes.pending, (state) => {
        state.status = 'loading';
        state.items = [];
      })
      .addCase(fetchShoes.fulfilled, (state, action) => {
        state.items = action.payload;
        state.status = 'success';
      })
      .addCase(fetchShoes.rejected, (state) => {
        state.status = 'error';
        state.items = [];
      });
  },
});

export const selectShoesData = (state) => state.shoes;

export const { setItems } = shoesSlice.actions;

export default shoesSlice.reducer;
```

Додаток В. cartSlice.js

```
import { createSlice } from '@reduxjs/toolkit';
import { getCartFromLS } from '../../utils/getCartFromLS';
import { calcTotalPrice } from '../../utils/calcTotalPrice';

const { items, totalPrice } = getCartFromLS();

const initialState = {
 totalPrice,
 items,
};

const cartSlice = createSlice({
 name: 'cart',
 initialState,
 reducers: {
  addItem(state, action) {
   const findItem = state.items.find((obj) => {
    return obj.id === action.payload.id && obj.size === action.payload.size;
   });
   if (findItem) {
    findItem.count++;
   } else {
    state.items.push({
     ...action.payload,
     count: 1,
    });
   }
   state.totalPrice = calcTotalPrice(state.items);
  },

  removeItem(state, action) {
   state.items = state.items.filter(
    (obj) => !(obj.id === action.payload.id && obj.size === action.payload.size),
   );
   state.totalPrice = calcTotalPrice(state.items);
  },

  clearItems(state) {
   state.items = [];
   state.totalPrice = 0;
  },

  minusItem(state, action) {
   const findItem = state.items.find((obj) => {
    return obj.id === action.payload.id && obj.size === action.payload.size;
   });
   console.log(findItem);
   if (findItem && findItem.count > 1) {
    findItem.count--;
    state.totalPrice = state.totalPrice - findItem.price;
   }
  },
 },
});

export const selectCart = (state) => state.cart;
export const selectCartItemById = (id) => (state) => state.cart.items.find((obj) => obj.id === id);
export const { addItem, removeItem, minusItem, clearItems } = cartSlice.actions;
```

```
export default cartSlice.reducer;
```

Додаток Г. filterSlice.js

```
import { createSlice } from '@reduxjs/toolkit';

const initialState = {
  searchValue: '',
  categoryId: 0,
  currentPage: 1,
  sort: {
    name: 'за популярністю',
    sortProperty: 'rating',
  },
};

const filterSlice = createSlice({
  name: 'filters',
  initialState,
  reducers: {
    setCategoryId(state, action) {
      state.categoryId = action.payload;
    },
    setSearchValue(state, action) {
      state.searchValue = action.payload;
    },
    setSort(state, action) {
      state.sort = action.payload;
    },
    setCurrentPage(state, action) {
      state.currentPage = action.payload;
    },
  },
});

export const selectFilter = (state) => state.filter;
export const selectSort = (state) => state.filter.sort;
export const { setCategoryId, setSort, setCurrentPage, setSearchValue } = filterSlice.actions;

export default filterSlice.reducer;
```

Додаток Ґ. store.js

```
import { configureStore } from '@reduxjs/toolkit';
import filter from './slices/filterSlice';
import cart from './slices/cartSlice';
import shoes from './slices/shoesSlice';

export const store = configureStore({
  reducer: {
    filter,
    cart,
    shoes,
  },
});
```

Додаток Д. index.js

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { BrowserRouter } from 'react-router-dom';
import App from './App';
import { store } from './redux/store';
import { Provider } from 'react-redux';

const root = ReactDOM.createRoot(document.getElementById('root'));

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

Додаток Е. Header.jsx

```
import React from 'react';
import logoSvg from '../assets/img/shoes-logo.svg';
import { Link, useLocation } from 'react-router-dom';
import { useSelector } from 'react-redux';

import Search from './Search';
import { selectCart } from '../redux/slices/cartSlice';

function Header() {
 const { items, totalPrice } = useSelector(selectCart);

 const location = useLocation();
 const isMounted = React.useRef(false);

 const totalCount = items.reduce((sum, item) => sum + item.count, 0);

 React.useEffect(() => {
  if (isMounted.current) {
   const json = JSON.stringify(items);
   localStorage.setItem('cart', json);
  }
  isMounted.current = true;
 }, [items]);

 const containsSubstring = (substring) => location.pathname.includes(substring);

 return (
  <div className="header">
   <div className="container">
    <Link to="/">
     <div className="header__logo">
      <img width="38" src={logoSvg} alt="Shoes Store logo" />
      <div>
       <h1>Shoes Store</h1>
       <p>Крокуй стильно!</p>
      </div>
     </div>
    </Link>
    {!containsSubstring('/cart') &&
     !containsSubstring('/shoes') &&
     !containsSubstring('/checkout') && <Search />}
    <div className="header__cart">
     {location.pathname !== '/cart' && (
      <Link to="/cart" className="button button--cart">
       <span>{totalPrice} ₴</span>
       <div className="button__delimiter"></div>
       <svg
        width="18"
        height="18"
        viewBox="0 0 18 18"
        fill="none"
        xmlns="http://www.w3.org/2000/svg">
        <path
         d="M6.33333 16.3333C7.06971 16.3333 7.66667 15.7364 7.66667 15C7.66667 14.2636 7.06971 13.6667 6.33333 13.6667C5.59695
13.6667 5 14.2636 5 15C5 15.7364 5.59695 16.3333 6.33333 16.3333Z"
         stroke="white"
         strokeWidth="1.8"
         strokeLinecap="round"
```

```
            strokeLinejoin="round"
           />
           <path
            d="M14.3333 16.3333C15.0697 16.3333 15.6667 15.7364 15.6667 15C15.6667 14.2636 15.0697 13.6667 14.3333 13.6667C13.597 13.6667 13 14.2636 13 15C13 15.7364 13.597 16.3333 14.3333 16.3333Z"
            stroke="white"
            strokeWidth="1.8"
            strokeLinecap="round"
            strokeLinejoin="round"
           />
           <path
            d="M4.78002 4.99999H16.3334L15.2134 10.5933C15.1524 10.9003 14.9854 11.176 14.7417 11.3722C14.4979 11.5684 14.1929 11.6727 13.88 11.6667H6.83335C6.50781 11.6694 6.1925 11.553 5.94689 11.3393C5.70128 11.1256 5.54233 10.8295 5.50002 10.5067L4.48669 2.82666C4.44466 2.50615 4.28764 2.21182 4.04482 1.99844C3.80201 1.78505 3.48994 1.66715 3.16669 1.66666H1.66669"
            stroke="white"
            strokeWidth="1.8"
            strokeLinecap="round"
            strokeLinejoin="round"
           />
          </svg>
          <span>{totalCount}</span>
         </Link>
        )}
       </div>
     </div>
    </div>
  );
}

export default Header;
```

Додаток Є. /Search/index.jsx

```
import React from 'react';
import debounce from 'lodash.debounce';

import searchIcon from './searchIcon.svg';
import clearIcon from './x_Icon.svg';
import styles from './Search.module.scss';

import { setSearchValue } from '../../redux/slices/filterSlice';
import { useDispatch } from 'react-redux';

const Search = () => {
  const dispatch = useDispatch();
  const [value, setValue] = React.useState('');

  const inputRef = React.useRef();

  const onClickClear = () => {
    dispatch(setSearchValue(''));
    setValue('');
    inputRef.current.focus();
  };

  const onMouseDownClear = (event) => {
    // фікс багу кнопки очистки поля пошуку
    event.preventDefault();
    onClickClear();
  };

  const onChangeInput = (event) => {
    setValue(event.target.value);
    updateSearchValue(event.target.value);
  };

  const updateSearchValue = React.useCallback(
    debounce((str) => {
      dispatch(setSearchValue(str));
    }, 300),
    [],
  );

  return (
    <div className={styles.root}>
      <img className={styles.icon} src={searchIcon} alt="Пошук"></img>
      <input
        ref={inputRef}
        value={value}
        onChange={onChangeInput}
        className={styles.input}
        placeholder="Знайти товар..."
      />
      {value && (
        <img
          onMouseDown={onMouseDownClear}
          onClick={onClickClear}
          className={styles.clearIcon}
          src={clearIcon}
          alt="Очистити поле пошуку"></img>
      )}
```

```
    </div>
  );
};

export default Search;
```

Додаток Ж. Footer.jsx

```jsx
import React from 'react';

function Footer() {
  return (
    <div className="footer">
      <div className="container">
        <p>Роботу виконав Білецький Дмитро. Всі права захищенні. СумДУ 2024</p>
      </div>
    </div>
  );
}

export default Footer;
```

Додаток 3. Home.jsx

```
import React from 'react';
import qs from 'qs';

import { useSelector, useDispatch } from 'react-redux';
import { useNavigate } from 'react-router-dom';
import { selectFilter, setCategoryId, setCurrentPage } from '../redux/slices/filterSlice';
import { fetchShoes, selectShoesData } from '../redux/slices/shoesSlice';

import Categories from '../components/Categories';
import Sort from '../components/Sort';
import ShoesBlock from '../components/ShoesBlock';
import Skeleton from '../components/ShoesBlock/Skeleton';
import Pagination from '../components/Pagination';

const Home = () => {
 const navigate = useNavigate();
 const dispatch = useDispatch();
 const { categoryId, sort, currentPage, searchValue } = useSelector(selectFilter);
 const { items, status } = useSelector(selectShoesData);

 const onChangeCategory = React.useCallback((id) => {
  dispatch(setCategoryId(id));
  dispatch(setCurrentPage(1));
 }, []);

 const onChangePage = (number) => {
  dispatch(setCurrentPage(number));
 };

 const [order, setOrder] = React.useState('desc');

 const getShoes = async () => {
  console.log(order);
  const sortBy = sort.sortProperty.replace('-', '');

  const category = categoryId > 0 ? `category=${categoryId}` : '';
  const search = searchValue ? `search=${searchValue}` : '';

  dispatch(
   fetchShoes({
    sortBy,
    order,
    category,
    search,
    currentPage,
   }),
  );
```

```
  window.scrollTo(0, 0);
};

React.useEffect(() => {
  getShoes();
}, [categoryId, sort.sortProperty, searchValue, currentPage, order]);

React.useEffect(() => {
  const queryString = qs.stringify({
    sortProperty: sort.sortProperty,
    categoryId,
    currentPage,
  });

  navigate(`?${queryString}`);
}, [categoryId, sort.sortProperty, currentPage]);

let docTitle = document.title;
window.addEventListener('blur', () => {
  document.title = 'Верніться будь ласка ◉_◉';
});
window.addEventListener('focus', () => {
  document.title = docTitle;
});

const shoes = items.map((obj) => <ShoesBlock {...obj} />);

const skeletons = [...new Array(8)].map((_, index) => <Skeleton key={index} />);

return (
  <div className="container">
    <div className="content__top">
      <Categories value={categoryId} onChangeCategory={onChangeCategory} />
      <Sort setOrder={setOrder} />
    </div>
    <h2 className="content__title">Каталог</h2>
    {status === 'error' && (
      <div className="content__error-info">
        <h2>Сталася помилка ☹</h2>
        <p>
          На жаль, не вдалося знайти моделі взуття. Перевірте правильність пошукового запиту або
          зверніться в службу підтримки.
        </p>
      </div>
    )}
    <div className="content__items">{status === 'loading' ? skeletons : shoes}</div>
    <Pagination currentPage={currentPage} onChangePage={onChangePage} />
  </div>
```

```
  );
};


export default Home;
```

Додаток И. /ShoesBlock/index.jsx

```
import React from 'react';

import { Link } from 'react-router-dom';

function ShoesBlock({ id, title, price, imageUrl }) {
 return (
  <div className="shoes-block-wrapper">
   <div className="shoes-block">
    <Link key={id} to={`/shoes/${id}`}>
     <img className="shoes-block__image" src={imageUrl} alt="Shoes" />
     <h4 className="shoes-block__title">{title}</h4>
     <div className="shoes-block__bottom">
      <div className="shoes-block__price">{price} ₴</div>
      <button className="button button--outline button--add">
       <span>Переглянути</span>
      </button>
     </div>
    </Link>
   </div>
  </div>
 );
}

export default ShoesBlock;
```

Додаток І. /ShoesBlock/Skeleton.jsx

```jsx
import React from 'react';
import ContentLoader from 'react-content-loader';

const Skeleton = (props) => (
  <ContentLoader
    speed={2}
    width={287}
    height={430}
    viewBox="0 0 287 430"
    backgroundColor="#f3f3f3"
    foregroundColor="#ecebeb"
    {...props}>
    <rect x="10" y="0" rx="0" ry="0" width="260" height="260" />
    <rect x="10" y="380" rx="0" ry="0" width="83" height="27" />
    <rect x="117" y="372" rx="20" ry="20" width="154" height="41" />
    <rect x="10" y="276" rx="0" ry="0" width="260" height="80" />
  </ContentLoader>
);

export default Skeleton;
```

Додаток Ї. /Pagination/index.jsx

```jsx
import React from 'react';
import ReactPaginate from 'react-paginate';
import styles from './Pagination.module.scss';

const Pagination = ({ currentPage, onChangePage }) => {
 return (
  <ReactPaginate
   className={styles.root}
   breakLabel="..."
   nextLabel=">"
   onPageChange={(event) => onChangePage(event.selected + 1)}
   pageRangeDisplayed={8}
   pageCount={3}
   forcePage={currentPage - 1}
   previousLabel="<"
   renderOnZeroPageCount={null}
  />
 );
};

export default Pagination;
```

Додаток Й. Categories.jsx

```jsx
import React from 'react';

function Categories({ value, onChangeCategory }) {
 const categories = ['Всі', 'Кросівки', 'Кеди', 'Черевики', 'Шльопанці'];

 return (
  <div className="categories">
   <ul>
    {categories.map((categoryName, i) => (
     <li key={i} onClick={() => onChangeCategory(i)} className={value === i ? 'active' : ''}>
      {categoryName}
     </li>
    ))}
   </ul>
  </div>
 );
}

export default Categories;
```

Додаток К. Sort/index.jsx

```
import React from 'react';
import { useSelector, useDispatch } from 'react-redux';
import { selectSort, setSort } from '../../redux/slices/filterSlice';

function Sort({ setOrder }) {
 const dispatch = useDispatch();
 const sort = useSelector(selectSort);
 const sortRef = React.useRef();

 const [open, setOpen] = React.useState(false);
 const list = [
  { name: 'за популярністю', sortProperty: 'rating' },
  { name: 'за ціною', sortProperty: 'price' },
  { name: 'за алфавітом ', sortProperty: 'title' },
 ];

 const onClickListItem = (obj) => {
  dispatch(setSort(obj));
  setOpen(false);
 };

 React.useEffect(() => {
  const handleClickOutside = (event) => {
   if (!event.composedPath().includes(sortRef.current)) {
    setOpen(false);
   }
  };

  document.body.addEventListener('click', handleClickOutside);

  return () => document.body.removeEventListener('click', handleClickOutside);
 }, []);

 return (
  <div ref={sortRef} className="sort">
   <div className="sort__label">
    <b>Сортувати:</b>
    <span onClick={() => setOpen(!open)}>{sort.name}</span>
    <button onClick={() => setOrder('asc')}>↑</button>
    <button onClick={() => setOrder('desc')}>↓</button>
   </div>
   {open && (
    <div className="sort__popup">
     <ul>
      {list.map((obj, i) => (
       <li
        key={i}
        onClick={() => onClickListItem(obj)}
        className={sort.sortProperty === obj.sortProperty ? 'active' : ''}>
        {obj.name}
       </li>
      ))}
     </ul>
    </div>
   )}
  </div>
 );
}
```

```
export default Sort;
```

Додаток Л. Product.jsx

```
import React from 'react';
import axios from 'axios';

import { useParams, useNavigate } from 'react-router-dom';
import { addItem } from '../redux/slices/cartSlice';
import { useDispatch } from 'react-redux';

import loading from '../assets/img/loading.gif';

const Product = () => {
 const [shoes, setShoes] = React.useState([]);
 const [activeSize, setActiveSize] = React.useState(0);
 const [isLoading, setIsLoading] = React.useState(true);

 const { id } = useParams();

 const navigate = useNavigate();
 const dispatch = useDispatch();

 React.useEffect(() => {
  async function fetchShoes() {
   try {
    const { data } = await axios.get('https://66264d33052332d55322633f.mockapi.io/items/' + id);
    setShoes(data);
    console.log(data);
   } catch (error) {
    alert('Помилка при отриманні товару!');
    navigate('/');
   } finally {
    setIsLoading(false);
   }
  }
  fetchShoes();
 }, []);

 if (!shoes) {
  return 'Загрузка...';
 }
 if (isLoading) {
  return (
   <div className="container">
    <img
     src={loading}
     style={{ width: '100px', height: '100px', display: 'block', margin: '0 auto' }}
     alt="Завантаження сторінки"
    />
   </div>
  );
 }
 const onClickAdd = () => {
  const { id, title, price, imageUrl, sizes } = shoes;
  const item = {
   id,
   title,
   price,
   imageUrl,
   size: sizes[activeSize],
  };
```

```
    dispatch(addItem(item));
  };

  return (
    <div className="container__product">
      <div className="image-container">
        <img src={shoes.imageUrl} alt="Зображення товару" />
      </div>
      <div className="info-container">
        <h1>{shoes.title}</h1>
        <h2>Ціна: {shoes.price} ₴</h2>
        <h3>Оберіть розмір:</h3>
        <div className="shoes-block">
          <div className="size-selector">
            <div className="shoes-block__selector">
              <ul>
                {shoes.sizes &&
                  shoes.sizes.map((size, i) => (
                    <li
                      key={size}
                      onClick={() => setActiveSize(i)}
                      className={activeSize === i ? 'active' : ''}>
                      {size}
                    </li>
                  ))}
              </ul>
            </div>
            <div className="shoes-block__bottom">
              <button
                onClick={() => onClickAdd(shoes)}
                className="button button--outline button--add">
                <svg
                  width="12"
                  height="12"
                  viewBox="0 0 12 12"
                  fill="none"
                  xmlns="http://www.w3.org/2000/svg">
                  <path
                    d="M10.8 4.8H7.2V1.2C7.2 0.5373 6.6627 0 6 0C5.3373 0 4.8 0.5373 4.8 1.2V4.8H1.2C0.5373 4.8 0 5.3373 0 6C0 6.6627 0.5373 7.2 1.2 7.2H4.8V10.8C4.8 11.4627 5.3373 12 6 12C6.6627 12 7.2 11.4627 7.2 10.8V7.2H10.8C11.4627 7.2 12 6.6627 12 6C12 5.3373 11.4627 4.8 10.8 4.8Z"
                    fill="white"
                  />
                </svg>
                <span>В кошик</span>
              </button>
            </div>
          </div>
        </div>
        <h3>Опис:</h3>
        <p>{shoes.description}</p>
      </div>
    </div>
  );
};

export default Product;
```

Додаток М. CartEmpty.jsx

```
import React from 'react';
import { Link } from 'react-router-dom';

import cartEmptyImg from '../assets/img/empty-cart.png';

const CartEmpty = () => {
 return (
  <>
   <div className="cart cart--empty">
    <h2>
     Кошик порожній <icon>😕</icon>
    </h2>
    <p>
     Ви ще не додали жодного товару в кошик.
     <br />
     Для того, щоб додати товари, перейдіть на головну сторінку.
    </p>
    <img src={cartEmptyImg} alt="Empty cart" />
    <Link to="/" className="button button--black">
     <span>Повернутися на головну</span>
    </Link>
   </div>
  </>
 );
};

export default CartEmpty;
```

Додаток Н. Cart.jsx

```
import React from 'react';
import { Link } from 'react-router-dom';
import { useDispatch, useSelector } from 'react-redux';
import { clearItems, selectCart } from '../redux/slices/cartSlice.js';
import CartItems from '../components/CartItems.jsx';
import CartEmpty from '../components/CartEmpty.jsx';

const Cart = () => {
 const dispatch = useDispatch();
 const { totalPrice, items } = useSelector(selectCart);

 const totalCount = items.reduce((sum, item) => sum + item.count, 0);

 const onClickClear = () => {
  if (window.confirm('Очистити корзину?')) dispatch(clearItems());
 };

 if (!totalCount) {
  return <CartEmpty />;
 }

 return (
  <div className="container container--cart">
   <div className="cart">
    <div className="cart__top">
     <h2 className="content__title">
      <svg
       width="18"
       height="18"
       viewBox="0 0 18 18"
       fill="none"
       xmlns="http://www.w3.org/2000/svg">
       <path
        d="M6.33333 16.3333C7.06971 16.3333 7.66667 15.7364 7.66667 15C7.66667 14.2636 7.06971 13.6667 6.33333 13.6667C5.59695 13.6667 5 14.2636 5 15C5 15.7364 5.59695 16.3333 6.33333 16.3333Z"
        stroke="white"
        stroke-width="1.8"
        stroke-linecap="round"
        stroke-linejoin="round"
       />
       <path
        d="M14.3333 16.3333C15.0697 16.3333 15.6667 15.7364 15.6667 15C15.6667 14.2636 15.0697 13.6667 14.3333 13.6667C13.597 13.6667 13 14.2636 13 15C13 15.7364 13.597 16.3333 14.3333 16.3333Z"
        stroke="white"
        stroke-width="1.8"
        stroke-linecap="round"
        stroke-linejoin="round"
       />
       <path
        d="M4.78002 4.99999H16.3334L15.2134 10.5933C15.1524 10.9003 14.9854 11.176 14.7417 11.3722C14.4979 11.5684 14.1929 11.6727 13.88 11.6667H6.83335C6.50781 11.6694 6.1925 11.553 5.94689 11.3393C5.70128 11.1256 5.54233 10.8295 5.50002 10.5067L4.48669 2.82666C4.44466 2.50615 4.28764 2.21182 4.04482 1.99844C3.80201 1.78505 3.48994 1.66715 3.16669 1.66666H1.66669"
        stroke="white"
        stroke-width="1.8"
        stroke-linecap="round"
        stroke-linejoin="round"
       />
      </svg>
      Кошик
```

```
      </h2>
      <div onClick={onClickClear} className="cart__clear">
       <svg
        width="20"
        height="20"
        viewBox="0 0 20 20"
        fill="none"
        xmlns="http://www.w3.org/2000/svg">
        <path
         d="M2.5 5H4.16667H17.5"
         stroke="#B6B6B6"
         stroke-width="1.2"
         stroke-linecap="round"
         stroke-linejoin="round"
        />
        <path
         d="M6.66663 5.00001V3.33334C6.66663 2.89131 6.84222 2.46739 7.15478 2.15483C7.46734 1.84227 7.89127 1.66667 8.33329
1.66667H11.6666C12.1087 1.66667 12.5326 1.84227 12.8451 2.15483C13.1577 2.46739 13.3333 2.89131 13.3333 3.33334V5.00001M15.8333
5.00001V16.6667C15.8333 17.1087 15.6577 17.5326 15.3451 17.8452C15.0326 18.1577 14.6087 18.3333 14.1666 18.3333H5.83329C5.39127
18.3333 4.96734 18.1577 4.65478 17.8452C4.34222 17.5326 4.16663 17.1087 4.16663 16.6667V5.00001H15.8333Z"
         stroke="#B6B6B6"
         stroke-width="1.2"
         stroke-linecap="round"
         stroke-linejoin="round"
        />
        <path
         d="M8.33337 9.16667V14.1667"
         stroke="#B6B6B6"
         stroke-width="1.2"
         stroke-linecap="round"
         stroke-linejoin="round"
        />
        <path
         d="M11.6666 9.16667V14.1667"
         stroke="#B6B6B6"
         stroke-width="1.2"
         stroke-linecap="round"
         stroke-linejoin="round"
        />
       </svg>
       <span>Очистити кошик</span>
      </div>
     </div>
     <div className="content__items">
      {items.map((item) => (
       <CartItems key={item.id} {...item} />
      ))}
     </div>
     <div className="cart__bottom">
      <div className="cart__bottom-details">
       <span>
        {' '}
        Всього товарів: <b>{totalCount}</b>{' '}
       </span>
       <span>
        {' '}
        Сума: <b>{totalPrice}₴</b>{' '}
       </span>
      </div>
      <div className="cart__bottom-buttons">
       <Link to="/" className="button button--outline button--add go-back-btn">
```

```
          <svg
            width="8"
            height="14"
            viewBox="0 0 8 14"
            fill="none"
            xmlns="http://www.w3.org/2000/svg">
            <path
              d="M7 13L1 6.93015L6.86175 1"
              stroke="#D3D3D3"
              stroke-width="1.5"
              stroke-linecap="round"
              stroke-linejoin="round"
            />
          </svg>
          <span>Повернутися назад</span>
        </Link>
        <Link to="/checkout">
          <div className="button pay-btn">
            <span>Оформити замовлення</span>
          </div>
        </Link>
      </div>
    </div>
  </div>
 );
};

export default Cart;
```

Додаток О. CartItems.jsx

```
import React from 'react';
import { useDispatch } from 'react-redux';
import { addItem, minusItem, removeItem } from '../redux/slices/cartSlice';

const CartItems = ({ id, title, size, price, count, imageUrl }) => {
 const dispatch = useDispatch();

 const onClickPlus = () => {
  dispatch(addItem({ id, size }));
 };

 const onClickMinus = () => {
  dispatch(minusItem({ id, size }));
 };

 const onClickRemove = () => {
  dispatch(removeItem({ id, size }));
 };

 return (
  <div class="cart__item">
   <div class="cart__item-img">
    <img class="shoes-block__image" src={imageUrl} alt="Shoes" />
   </div>
   <div class="cart__item-info">
    <h3>{title}</h3>
    <p>Розмір: {size}</p>
   </div>
   <div class="cart__item-count">
    <div
     onClick={onClickMinus}
     class="button button--outline button--circle cart__item-count-minus">
     <svg
      width="10"
      height="10"
      viewBox="0 0 10 10"
      fill="none"
      xmlns="http://www.w3.org/2000/svg">
      <path
       d="M5.92001 3.84V5.76V8.64C5.92001 9.17016 5.49017 9.6 4.96001 9.6C4.42985 9.6 4.00001 9.17016 4.00001 8.64L4 5.76L4.00001 3.84V0.96C4.00001 0.42984 4.42985 0 4.96001 0C5.49017 0 5.92001 0.42984 5.92001 0.96V3.84Z"
       fill="#EB5A1E"
      />
      <path
       d="M5.75998 5.92001L3.83998 5.92001L0.959977 5.92001C0.429817 5.92001 -2.29533e-05 5.49017 -2.29301e-05 4.96001C-2.2907e-05 4.42985 0.429817 4.00001 0.959977 4.00001L3.83998 4L5.75998 4.00001L8.63998 4.00001C9.17014 4.00001 9.59998 4.42985 9.59998 4.96001C9.59998 5.49017 9.17014 5.92001 8.63998 5.92001L5.75998 5.92001Z"
       fill="#EB5A1E"
      />
     </svg>
    </div>
    <b>{count}</b>
    <div
     onClick={onClickPlus}
     class="button button--outline button--circle cart__item-count-plus">
     <svg
      width="10"
      height="10"
      viewBox="0 0 10 10"
```

```
        fill="none"
        xmlns="http://www.w3.org/2000/svg">
        <path
         d="M5.92001 3.84V5.76V8.64C5.92001 9.17016 5.49017 9.6 4.96001 9.6C4.42985 9.6 4.00001 9.17016 4.00001 8.64L4 5.76L4.00001
3.84V0.96C4.00001 0.42984 4.42985 0 4.96001 0C5.49017 0 5.92001 0.42984 5.92001 0.96V3.84Z"
         fill="#EB5A1E"
        />
        <path
         d="M5.75998 5.92001L3.83998 5.92001L0.959977 5.92001C0.429817 5.92001 -2.29533e-05 5.49017 -2.29301e-05 4.96001C-2.2907e-05
4.42985 0.429817 4.00001 0.959977 4.00001L3.83998 4L5.75998 4.00001L8.63998 4.00001C9.17014 4.00001 9.59998 4.42985 9.59998
4.96001C9.59998 5.49017 9.17014 5.92001 8.63998 5.92001L5.75998 5.92001Z"
         fill="#EB5A1E"
        />
       </svg>
      </div>
     </div>
     <div class="cart__item-price">
      <b>{price * count} ₴</b>
     </div>
     <div class="cart__item-remove">
      <div onClick={onClickRemove} class="button button--outline button--circle">
       <svg
        width="10"
        height="10"
        viewBox="0 0 10 10"
        fill="none"
        xmlns="http://www.w3.org/2000/svg">
        <path
         d="M5.92001 3.84V5.76V8.64C5.92001 9.17016 5.49017 9.6 4.96001 9.6C4.42985 9.6 4.00001 9.17016 4.00001 8.64L4 5.76L4.00001
3.84V0.96C4.00001 0.42984 4.42985 0 4.96001 0C5.49017 0 5.92001 0.42984 5.92001 0.96V3.84Z"
         fill="#EB5A1E"
        />
        <path
         d="M5.75998 5.92001L3.83998 5.92001L0.959977 5.92001C0.429817 5.92001 -2.29533e-05 5.49017 -2.29301e-05 4.96001C-2.2907e-05
4.42985 0.429817 4.00001 0.959977 4.00001L3.83998 4L5.75998 4.00001L8.63998 4.00001C9.17014 4.00001 9.59998 4.42985 9.59998
4.96001C9.59998 5.49017 9.17014 5.92001 8.63998 5.92001L5.75998 5.92001Z"
         fill="#EB5A1E"
        />
       </svg>
      </div>
     </div>
    </div>
  );
};

export default CartItems;
```

Додаток П. /NotFoundBlock/index.jsx

```jsx
import React from 'react';
import styles from './NotFoundBlock.module.scss';

const NotFoundBlock = () => {
 return (
  <div className={styles.root}>
   <h1>
    <span>☹</span>
    <br />
    Нічого не знайдено
   </h1>
   <p className={styles.description}>На жаль, така сторінка в нашому магазині відсутня</p>
  </div>
 );
};

export default NotFoundBlock;
```

Додаток Р. NotFound.jsx

```jsx
import React from 'react';
import NotFoundBlock from '../components/NotFoundBlock';

const NotFound = () => {
  return <NotFoundBlock />;
};

export default NotFound;
```

Додаток С. App.js

```
import React from 'react';
import './scss/app.scss';
import { Routes, Route } from 'react-router-dom';

import Home from './pages/Home';
import NotFound from './pages/NotFound';
import Cart from './pages/Cart';
import Product from './pages/Product';
import MainLayout from './layouts/MainLayout';
import CheckOut from './components/CheckOut';

function App() {
 return (
  <Routes>
   <Route path="/" element={<MainLayout />}>
    <Route path="" element={<Home />} />
    <Route path="cart" element={<Cart />} />
    <Route path="shoes/:id" element={<Product />} />
    <Route path="checkout" element={<CheckOut />} />
    <Route path="*" element={<NotFound />} />
   </Route>
  </Routes>
 );
}

export default App;
```