ДВНЗ «Донецький національний технічний університет»

Факультет комп'ютерно-інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету)

Кафедра прикладної математики та інформатики
(повна назва кафедри)

«До захисту допущено»
Зав. кафедри ПМІ

_____ Наталія МАСЛОВА
(підпис) (ім'я та прізвище)

"___"_____20__ р.

Випускна кваліфікаційна робота

бакалавра
(освітній ступінь)

на тему: Прикладні аспекти створення інтернет-магазину

зі спецчастиною: Розробка веб-додатку інтернет-магазину взуття

Виконав (-ла): студент (-ка) _4_ курсу, групи ____ІПЗ-20_____
(шифр групи)

спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)

освітньої програми _____

_____ Кузьменко Дмитро Едуардович _____ _____
(прізвище та ініціали) (підпис)

Керівник доц. ПМІ, к.ф.м.н, доц., Олена ЛЮБИМЕНКО _____.
(посада, науковий ступінь, вчене звання, ім'я та прізвище) (підпис)

Рецензент доцент каф. ЕТіКІ, к.т.н., доц. Олександр ШТЕПА _____
(посада, науковий ступінь, вчене звання, ім'я та прізвище) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Луцьк – 2024

ДВНЗ «Донецький національний технічний університет»
(назва вищого навчального закладу)

Факультет комп'ютерно-інформаційних технологій та автоматизації
Кафедра Прикладної математики та інформатики
Освітній ступень Бакалавр
Спеціальність 121 Інженерія програмного забезпечення
Освітня програма 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Зав. кафедри ПМІ
_____ /Н. Маслова/
«____»_____2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТУ

Кузьменку Дмитру Едуардовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Прикладні аспекти створення інтернет-магазину»

Спецчастина «Розробка веб-додатку інтернет-магазину взуття»

Керівник роботи Любименко О.М., доц., к.ф.-м.н., доцент кафедри ПМІ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом від « « 25» квітня 2024 року № 164
2. Строк подання студентом роботи 09 червня 2024 року
3. Вихідні дані до роботи результати переддипломної практики

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1) Обґрунтування вибору методу розробки та постановка задач дослідження
2) Розробка методу вибору спеціальності, моделі та алгоритмів роботи інформаційної системи
3) Розробка та структура інтернет магазину
4) Розробка математичних моделей
5) Тестування роботи додатку

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
робота містить 1 таблиць, 45 рисунків, 9 слайдів презентації та інший графічний матеріал, у кількості, достатній для демонстрації сутності роботи

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Назарова І.А.		

7.Дата видачі завдання 25.04.2024

КАЛЕНДАРНИЙ ПЛАН

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів виконання роботи	Строк виконання етапів роботи	Примітки
1	Аналіз стану систем для інтернет магазину	15.04.2024-20.04.2024	
2	Розробка структури інтернет магазину	21.04.2024-24.04.2024	
3	Розробка програмного забезпечення	25.04.24-30.04.24	
4	Розробка математичних моделей	01.05.24-05.05.24	
5	Тестування	06.05.24 – 25.05.24	
5	Оформлення пояснювальної записки	26.05.24-09.06.24	
6	Здача та захист роботи	За графіком	

Студент_____ Дмитро КУЗЬМЕНКО
(підпис)

Керівник _____ Олена ЛЮБИМЕНКО
(підпис) (посада, ПІБ)

АНОТАЦІЯ

Дмитро КУЗЬМЕНКО. Прикладні аспекти створення інтернет-магазину / Випускна кваліфікаційна робота на здобуття освітньо-кваліфікаційного рівня «бакалавр» за спеціальністю 121 Інженерія програмного забезпечення. ДВНЗ ДонНТУ, Луцьк, 2024.

Об’єктом дослідження є весь процес розробки програмного забезпечення для магазину взуття, починаючи від концепції і завершуючи впровадженням і підтримкою.

Предметом дослідження є аспекти розробки програмного забезпечення для магазину взуття, включаючи аналіз потреб користувачів, методи взаємодії з програмним забезпеченням.

Метою дослідження є розробка ефективного та зручного програмного забезпечення для магазину взуття, яке забезпечить покращений користувацький досвід при здійсненні покупок онлайн. Особлива увага приділяється створенню нових алгоритмів взаємодії з користувачем, що підвищать зручність і доступність процесу вибору та придбання взуття.

Методи дослідження включають аналіз потреб користувачів, проектування користувацького інтерфейсу, розробку програмного забезпечення, функціональне та зручне тестування, впровадження системи, технічна підтримку та регулярне оновлення.

Наукова новизна полягає в розробці нових алгоритмів взаємодії з користувачем для покращення його досвіду покупок онлайн.

Практична цінність полягає в можливості створення зручного та ефективного онлайн середовища для покупок взуття, яке може бути використане покупцями з будь-якого місця та в будь-який час. Розроблений додаток може підвищити зручність та доступність покупок, а також зробити процес вибору і придбання взуття більш ефективним для клієнтів.

ІНТЕРНЕТ-МАГАЗИН, REACT, PYTHON, ІНФОРМАЦІЙНА СИСТЕМА, ОПТИМІЗАЦІЯ ПОКУПКИ, ВЕБ-ІНТЕРФЕЙС.

ABSTRACT

Dmytro KUZMENKO. Applied aspects of creating an online store / Graduation qualification work for obtaining the educational and qualification level "bachelor" in the specialty 121 Software engineering. - DVNZ DonNTU, Lutsk, 2024.

The object of the study is the entire process of software development for a shoe store, starting from the concept and ending with implementation and support.

The subject of the study is the aspects of software development for a shoe store, including the analysis of user needs, methods of interaction with the software.

The purpose of the research is to develop an efficient and user-friendly software for a shoe store that will provide an improved user experience when shopping online. Special attention is paid to the creation of new user interaction algorithms that will increase the convenience and accessibility of the process of choosing and purchasing shoes.

Research methods include user needs analysis, user interface design, software development, functional and usability testing, system implementation, technical support, and regular updates.

Scientific novelty consists in the development of new algorithms of interaction with the user to improve his online shopping experience.

The practical value lies in the ability to create a convenient and efficient online environment for shopping shoes that can be used by buyers from anywhere and at any time. The developed application can increase the convenience and accessibility of shopping, as well as make the process of choosing and purchasing shoes more efficient for customers.

ONLINE STORE, REACT, PYTHON, INFORMATION SYSTEM, PURCHASE OPTIMIZATION, WEB INTERFACE.

ЗМІСТ

ВСТУП 8

1 ОБГРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ 11

1.1 Аналіз стану інтернет магазинів для покупки взуття 11

1.2 Порівняльний аналіз аналогів 11

1.3 Аналіз методів розв’язання задачі 16

1.4 Постановка задачі 17

1.5 Висновки за розділом 17

2 РОЗРОБКА МЕТОДУ ВИБОРУ СПЕЦІАЛЬНОСТІ, МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ ІНФОРМАЦІЙНОЇ СИСТЕМИ 18

2.1 Розробка методу та алгоритмів роботи системи 18

2.2 Розробка веб-інтерфейсу користувача 20

2.3 Висновки за розділом 22

3 РОЗРОБКА ТА СТРУКТУРА ІНТЕРНЕТ МАГАЗИНУ 23

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного засобу. 23

3.2 MVC – архітектурний шаблон додатку 26

3.3 Фреймворк Django для розробки веб-додатків 28

3.4 Фреймворк React для розробки веб-інтерфейсу 31

3.5 Основна структура клієнтської частини додатку 32

3.6 Огляд компоненту Basket 35

3.7 Огляд компоненту About 37

3.8 Огляд компоненту Contacts 38

3.9 Огляд компонентів Registration та Login 39

3.10 Висновки за розділом 41
4 РОЗРОБКА МАТЕМАТИЧНИХ МОДЕЛЕЙ 42
4.1 Математична модель кількості відвідувачів у різні сезони року 42
4.2 Математична модель «трафік і поведінка користувачів» 46
4.3 Математична модель розподілу навантаження 49
4.4 Математична модель аналізу користувацької поведінки 52
4.5 Математична модель швидкості завантаження 55
4.5.1 Модель вимірювання початкового часу для завантаження комопоненту «Home» 55
4.5.2 Модель «швидкості пошуку» 57
4.5.3 Модель швидкості авторизації 58
4.6 Висновки за розділом 59
5 ТЕСТУВАННЯ 60
5.1 Тестування категорій 60
5.2 Тестування покупки товарів 61
5.3 Тестування пошуку товарів 64
5.4 Тестування реєстрації та авторизації 65
5.5 Висновок за розділом 67
ВИСНОВКИ 69
ПЕРЕЛІК ПОСИЛАНЬ 72
ДОДАТОК А Зауваження нормоконтролера 74
ДОДАТОК Б Текст програмного застосунку 75

ВСТУП

В умовах сучасного ринку взуття інтернет-магазини стають домінуючими у здійсненні покупок через їх зручність та доступність. Зростання конкуренції в цьому сегменті вимагає постійного вдосконалення процесу покупки та підвищення його ефективності.

Інформаційні технології стали необхідним елементом сучасного бізнесу, а їх вплив на ефективність управління та прийняття рішень стає все більш значущим. Застосування інформаційної системи для інтернет-магазину дозволить оптимізувати процес покупки, забезпечити зручний та інтуїтивно зрозумілий інтерфейс для користувачів, а також здійснювати аналіз даних для підвищення ефективності бізнесу.

Розробка інформаційної системи для інтернет-магазину взуття має на меті покращення взаємодії з клієнтами, забезпечення зручності та швидкості у виборі та замовленні товарів, а також підвищення конкурентоспроможності підприємства на ринку. Аналіз та обробка даних про попит, покупки та поведінку клієнтів дозволить підприємству приймати обґрунтовані рішення щодо асортименту, ціноутворення та маркетингових стратегій.

Таким чином, розробка інформаційної системи для інтернет-магазину взуття є актуальною та перспективною задачею, яка сприятиме підвищенню ефективності бізнесу та задоволенню потреб сучасних споживачів.

Мета бакалаврської дипломної роботи є розробка та впровадження інформаційної системи, що дозволить через інтернет-магазин спростити вибір взуття для покупців та оптимізувати процес прийняття рішень.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Розробити методику рекомендацій взуття, яка враховуватиме індивідуальні потреби та вподобання клієнтів.

2. Розробити модель системи пошуку взуття, що базується на введених користувачем параметрах.

3. Створити інтуїтивно зрозумілий і зручний інтерфейс для користувачів інформаційної системи, що дозволить легко та швидко знаходити потрібне взуття.

4. Реалізувати програмний продукт - інформаційну систему для інтернет-магазину взуття на базі обраної технологічної стеку.

5. Провести тестування розробленої системи для визначення її ефективності та коректності роботи, а також збору фідбеку від користувачів.

Об’єкт дослідження – процес розробки інформаційної системи для вибору взуття у інтернет магазинах.

Предмет дослідження – методи та засоби розробки інформаційних системта веб-інтерфейсів для покращення вибору взуття в інтернет-магазинах.

У процесі досліджень використовувались:

- методи прогнозування для системи прийняття рішень;

- методи теорії алгоритмів для побудови алгоритмів роботи програми;

- методи створення графічних зображень для візуального оформлення інтерфейсу;

- методи тестування для підтвердження працездатності системи та її відповідності заданим вимогам.

У даному дослідженні досягнуто значного прогресу в розвитку методів статистичного аналізу для вибору взуття в інтернет-магазинах. Особливо значущим є вдосконалення методу багатокритеріального аналізу, який реалізує аналіз спеціальностей враховуючи вказані користувачем параметри та їх пріоритети.

Цей підхід дозволяє не лише враховувати більш широкий спектр факторів при виборі взуття, але й надає користувачам можливість визначитися з вибором на основі їхніх індивідуальних уподобань та потреб. Такий підхід є новаторським у контексті існуючих методів аналізу вибору товарів в інтернеті

та може мати значний вплив на покращення досвіду покупок клієнтів у веб-магазинах.

Практична цінність отриманих результатів полягає в реалізації інформаційної системи для вибору взуття у інтернет-магазинах, яка може використовуватись користувачами для зручного та ефективного вибору взуття.

1 ОБГРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану інтернет магазинів для покупки взуття

У сучасних умовах розвитку в Україні ринкових відносин, вибір взуття в інтернет-магазинах стає важливою складовою купівельного процесу, який має значний вплив на якість покупок, рівень зручності та задоволення споживача.

Як відомо, успішний вибір взуття може відображати стиль, особисті уподобання та впливати на комфорт під час носіння. При цьому невірний вибір може призвести до незручностей, непридатності взуття та не задоволення від покупки.

У даному контексті актуальною стає розробка інформаційної системи, яка допоможе споживачам приймати обґрунтовані рішення під час вибору взуття в інтернет-магазинах. Ця система надасть користувачам зручний інструмент для перегляду різноманітних варіантів взуття, аналізу їх характеристик та вибору найбільш підходящого варіанту.

Отже, розроблена інформаційна система має велику практичну цінність для споживачів, допомагаючи їм зробити свідомий вибір під час покупки взуття в інтернет-магазинах, тим самим підвищуючи задоволення від купівлі та комфорт у подальшому використанні.

1.2 Порівняльний аналіз аналогів

На сьогоднішній день існує декілька сервісів для пошуку взуття в інтернет-магазинах, з них найвідомішими є:

- Rozetka.ua;
- Sezon.ua;
- Foxtrot.ua.

Розетка.ua - один із найбільших і найвідоміших інтернет-магазинів в

Україні, який спеціалізується на продажі різноманітних товарів, включаючи взуття. Інтерфейс Rozetka.ua наведено на рис.1.1 [1].

Переваги:

- Зручний пошук: Сайт магазину має зручний інтерфейс для пошуку взуття за різними критеріями, такими як розмір, колір, стиль тощо, що дозволяє швидко знайти потрібний товар.

- Доступні ціни: Розетка.ua часто пропонує знижки та акції на взуття, що робить покупки більш доступними для клієнтів.

- Якість обслуговування: Магазин славиться своєю високою якістю обслуговування, швидкими та надійними доставками, а також зручними способами оплати.

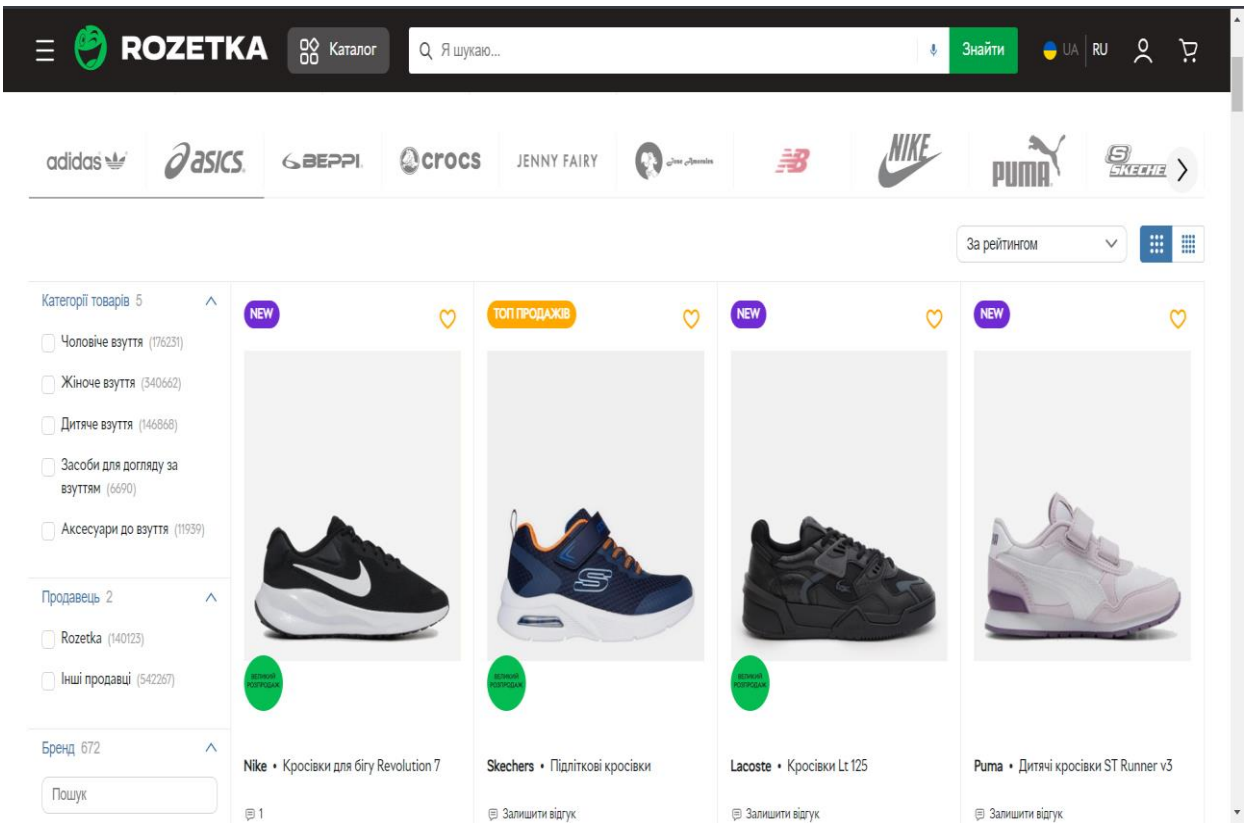

Рисунок 1.1 – Інтерфейс Rozetka.ua

Недоліки:

- Обмежена доступність товарів: Хоча асортимент Розетка.ua є великим, але деякі моделі взуття можуть бути обмеженими в наявності або недоступними для покупки в певні періоди часу.

- Можливість отримання неякісного товару: Як і в інших інтернет-магазинах, існує ризик отримання товару, який не відповідає опису або має дефекти. В такому випадку може знадобитися повернення товару або обмін, що може бути не зручно для деяких покупців.

- Вартість доставки: Доставка товарів може бути додатковими витратами для покупця, особливо якщо він знаходиться віддалено від центральних регіонів або потребує швидкої доставки.

- Обмежені можливості примірки: Одним із недоліків інтернет-магазинів загалом є відсутність можливості примірки взуття перед покупкою, що може бути проблемою для тих, хто прагне переконатися у відповідності розміру та комфорту до покупки.

Сайт Sezon.ua - це інтернет-магазин, спеціалізований на продажу взуття та аксесуарів в Україні. Він пропонує широкий асортимент товарів від провідних світових брендів і власних торгових марок. Інтерфейс Sezon.ua наведено на рис.1.2 [2].

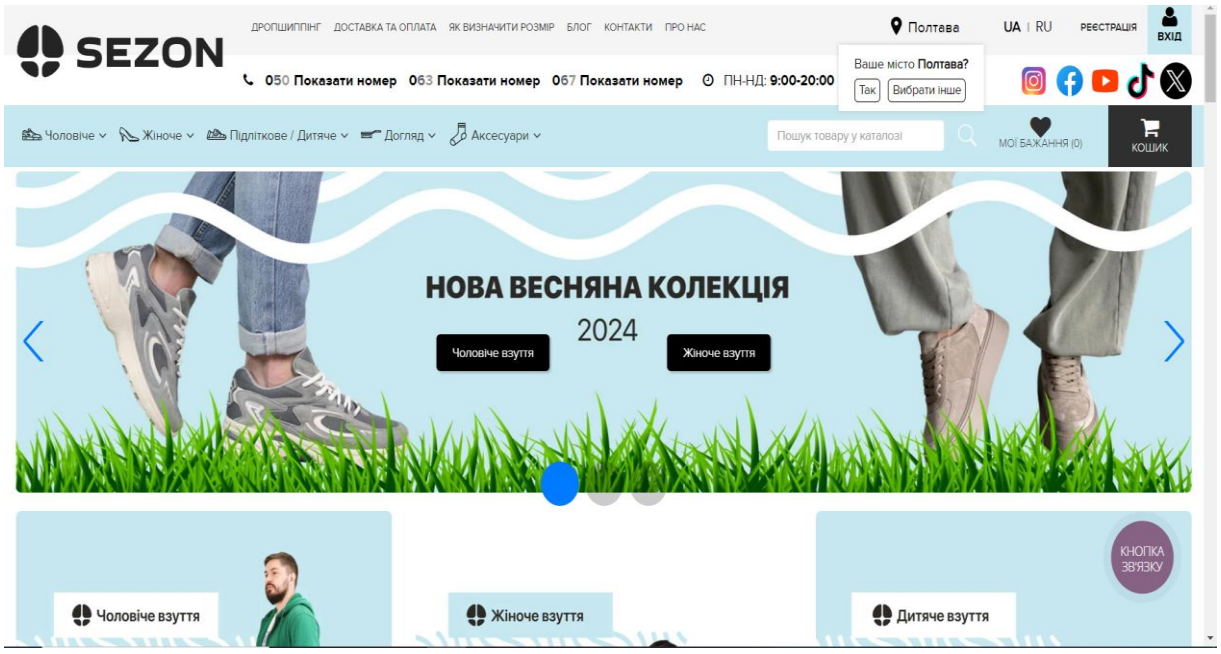

Рисунок 1.2 – Сайт Sezon.ua

Переваги:

- Широкий вибір: Сайт пропонує великий асортимент взуття для всієї родини - від кросівок та кедів до ботфортів та чобітків, що дозволяє знайти відповідний варіант для будь-якого сезону та стилю.

- Акції та знижки: Sezon.ua часто проводить акції та надає знижки на взуття, що робить покупки більш доступними для клієнтів.

- Зручний пошук: Сайт має зручний інтерфейс для пошуку взуття за різними параметрами, такими як розмір, категорія, ціновий діапазон тощо, що полегшує процес вибору товару.

- Доставка та оплата: Sezon.ua пропонує зручні способи доставки та оплати замовлення, включаючи можливість оплати після отримання товару.

- Якість товарів: Багато покупців відзначають високу якість взуття, яке продається на сайті Sezon.ua, та задоволеністю з їхніми покупками.

Недоліки:

- Обмежений асортимент.

- Висока вартість доставки.

Фокстрот - це один із найбільших інтернет-магазинів техніки та електроніки в Україні Інтерфейс Foxtrot.ua наведено на рис.1.3 [3].

Переваги:

1. Широкий асортимент товарів: Фокстрот пропонує великий вибір продукції від провідних світових брендів у таких категоріях, як смартфони, комп'ютери, побутова техніка, електроніка, планшети та ігрові консолі.

2. Доступні ціни: Магазин Фокстрот пропонує конкурентоспроможні ціни на багато товарів, а також регулярні акції та знижки для своїх клієнтів.

3. Зручність покупок: Клієнти можуть замовляти товари через інтернет зручним для них способом і отримувати їх у найближчому магазині Фокстрот або відділенні служби доставки.

4. Гарантія якості: Фокстрот гарантує якість своїх товарів та надає покупцям можливість обміну або повернення товару відповідно до законодавства.

5. Професійне обслуговування: Команда Фокстрот надає професійну

консультацію та допомогу клієнтам у виборі товарів, а також після продажне обслуговування.

Недоліки:

1. Можливість затримок у доставці: У зв'язку з різними обставинами, такими як погодні умови чи транспортні проблеми, можуть виникати затримки у доставці замовлень.

2. Обмеженість асортименту в магазинах: У деяких магазинах Фокстрот може бути обмежений вибір товарів порівняно з онлайн-платформою.

3. Потреба в реєстрації: Для здійснення покупок в інтернет-магазині Фокстрот потрібно створити обліковий запис, що може бути не зручно для деяких користувачів.

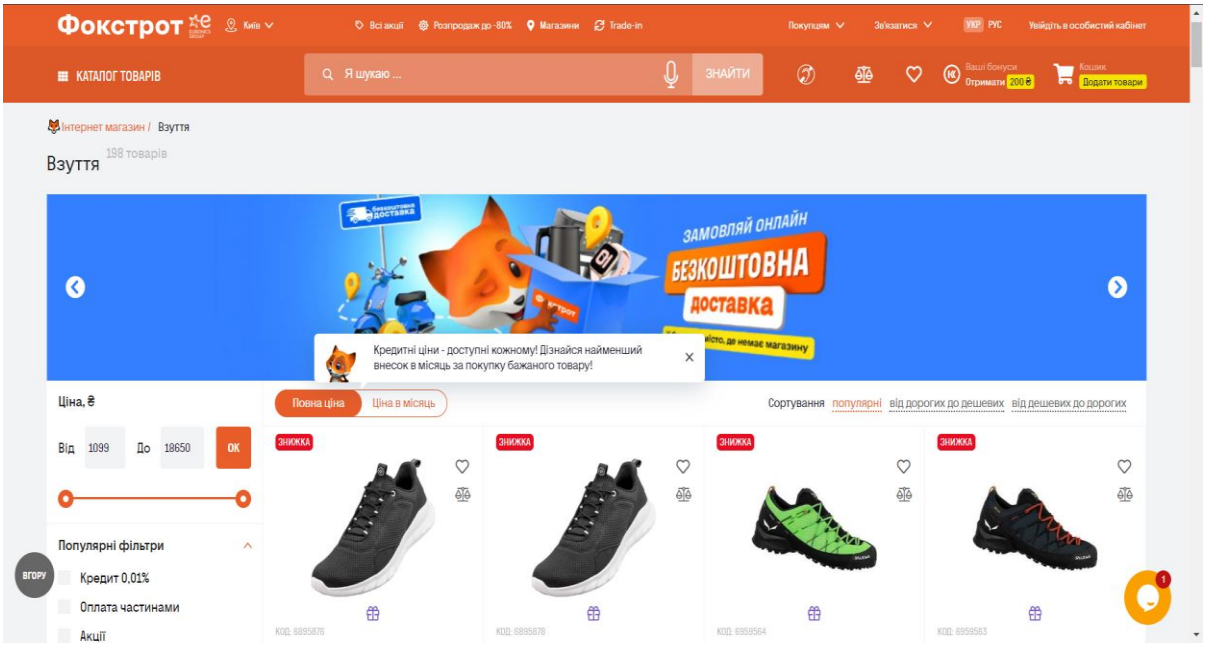

Рисунок 1.3 – Сайт Foxtrot.ua

Порівняємо всі системи між собою, результати зведемо в таблицю 1.1.

Таблиця 1.1 – Порівняльна характеристика сервісів для пошуку закладів освіти

Критерії	Rozetka	Sezon.ua	Foxtrot.ua	Власна система
Зручний інтерфейс	+	+	-	+
Доступні ціни	+	-	+	+
Широкий асортимент товарів	-	+	+	-
Зручність покупок	-	-	+	+
Зручний пошук	+	+	-	+
Загальна оцінка	60%	60%	60%	80%

Відповідно до порівняльних характеристик розробка власної системи для покупки взуття є доцільною. Такий програмний продукт зможе покрити недоліки існуючих рішень та забезпечити новий функціонал.

1.3 Аналіз методів розв’язання задачі

Для вирішення задачі вибору взуття існує кілька методів. Найшвидшим з них є розробка окремого модуля, який дозволить клієнтам отримати рекомендації за їхніми перевагами та стилем. Цей підхід може бути ефективним, але вимагає додаткових зусиль з боку розробників веб-інтерфейсу. Недоліком є те, що частина роботи буде покладена на розробників веб-інтерфейсу, що інтегрують модуль. Тому цей метод може бути менш бажаним для реалізації.

Кращим варіантом є розробка власної інформаційної системи для вибору взуття. Цей підхід дозволить одразу розробити модуль та інтерфейс до нього, спрощуючи процес розробки. Однак цей метод може бути складнішим через застосування різних підходів одночасно.

Враховуючи переваги і недоліки кожного методу, було прийнято рішення використати метод розробки власної інформаційної системи з

влаштованим модулем вибору взуття, що дозволить більш повноцінно та ефективно розробити програмний продукт.

1.4 Постановка задачі

Завданням бакалаврської дипломної роботи є розробка інтернет магазин взуття.

Система буде створюватись з використанням мови програмування Java Script, Python та їх фреймворків.

Для розробки інформаційної системи для інтернет-магазину взуття потрібно вирішити наступні завдання:

- Створити модель системи підбору взуття на основі введених користувачем параметрів, яка буде рекомендувати оптимальні варіанти взуття.
- Розробити інтерфейс та дизайн інформаційної системи, що буде зрозумілим та зручним у використанні навіть для непідготовлених користувачів.
- Програмно реалізувати спеціалізовану інформаційну систему для вибору взуття у магазині, яка буде ефективно працювати з базою даних взуття та забезпечувати швидкий та точний підбір варіантів для клієнтів.
- Провести тестування програмного продукту для перевірки його працездатності, безпеки та відповідності вимогам.

1.5 Висновки за розділом

У першому розділі було розглянуто стан сучасних інтернет магазинів. Було проведено порівняння з Rozetka.ua, Sezon.ua та Foxtrot.ua, у результаті якого було визначено,що існуючі сервіси мають певні вади. Таким чином було доведено доцільність розробки власного програмного рішення. На основі отриманої інформації було сформовано перелік задач, які необхідно виконати для розробки власного інтернет магазину.

2 РОЗРОБКА МЕТОДУ ВИБОРУ СПЕЦІАЛЬНОСТІ, МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Розробка методу та алгоритмів роботи системи

Для розробки інформаційної системи інтернет-магазину взуття необхідно розробити метод пошуку товарів та алгоритми роботи системи, які враховують потреби користувачів. Основні кроки методу пошуку та алгоритмів роботи системи такі:

1. Пошук товарів: Користувач отримує доступ до списку всіх товарів, що доступні в інтернет-магазині. Можливість фільтрації товарів за різними параметрами дозволяє користувачеві знайти потрібний товар швидко та ефективно.

2. Фільтрація результатів: Після введення необхідних параметрів фільтрації, користувач може обрати, як саме він хоче відфільтрувати товари: переглянути всі доступні товари, застосувати фільтри для пошуку конкретного товару.

3. Вибір товару та оформлення замовлення: Після ознайомлення з результатами пошуку, користувач може вибрати потрібний товар та перейти до оформлення замовлення.

Алгоритм роботи методу вибору взуття в інтернет-магазині наведено нижче (рис. 2.1). Розглянувши цей алгоритм, можна зробити висновок, що запропонований підхід дозволяє користувачам ефективно шукати та вибирати потрібну взуття, спрощуючи процес покупки.

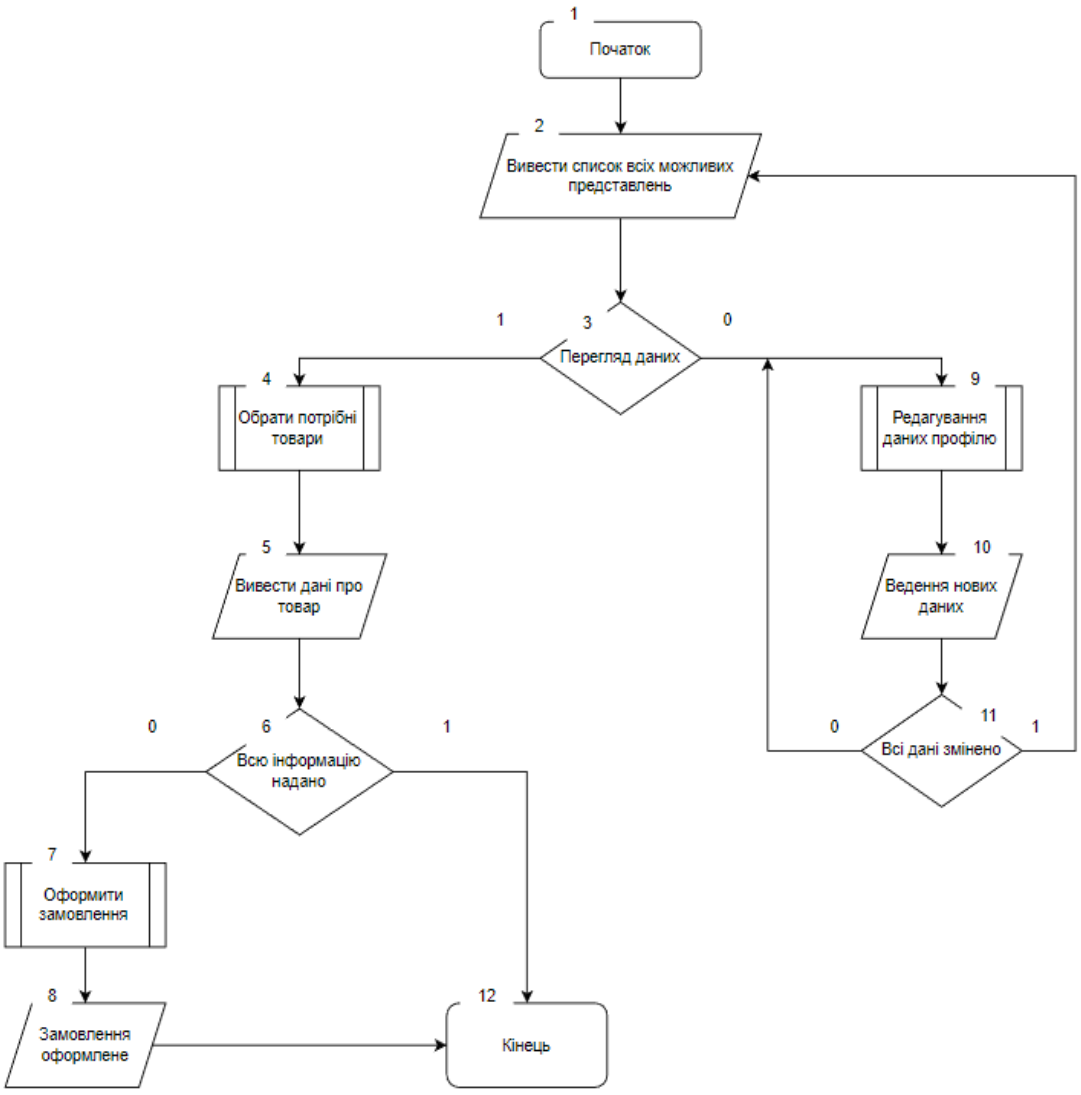

Рисунок 2.1 – Блок-схема загального алгоритму роботи системи

Для зрозуміння роботи методу відбору взуття було розроблено окремий алгоритм (див. рисунок 2.2). Згідно з цим алгоритмом, користувач спочатку визначає свої вимоги до взуття, такі як тип (чоботи, кросівки, туфлі тощо) і інші. Система аналізує ці дані, а потім шукає серед асортименту товарів ті, що найбільш відповідають вказаним критеріям.

Після цього результат виводиться на екран у вигляді переліку рекомендованих товарів, де користувач може ознайомитися з різними параметрами обраного взуття. Також система може рекомендувати подібні моделі, які можуть зацікавити користувача.

В цей спосіб користувач має змогу швидко та зручно знайти потрібну пару взуття, враховуючи його власні вимоги та вподобання.

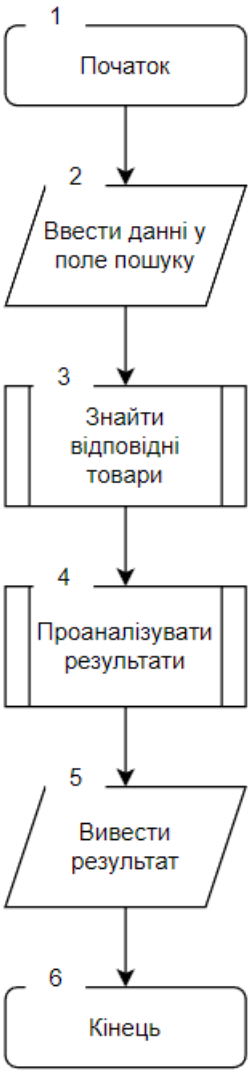

Рисунок 2.2 – Блок-схема алгоритму методу пошуку товарів

2.2 Розробка веб-інтерфейсу користувача

Основними критеріями при розробці інтерфейсу були простота та зручність. Для зручності в навігації сайту в шапку були додані кнопки «Пошук», «Головна», «Про нас», «Контакти», «Увійти» (рис. 2.3).

Рисунок 2.3 – Шапка сайту

Для зручності перегляду всі дані були розбиті на категорії (рисунок 2.4). Таким чином користувачеві не потрібно довго розшукувати необхідну йому інформацію. Аналогічно були розділення представлення даних кожної категорії (рис. 2.5).

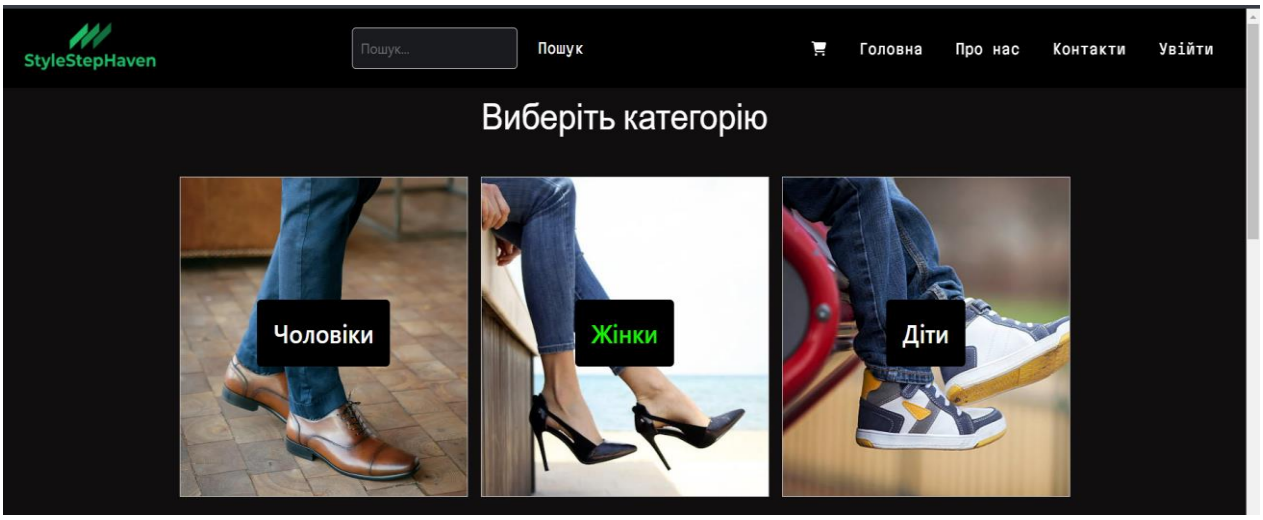

Рисунок 2.4 – Список категорій

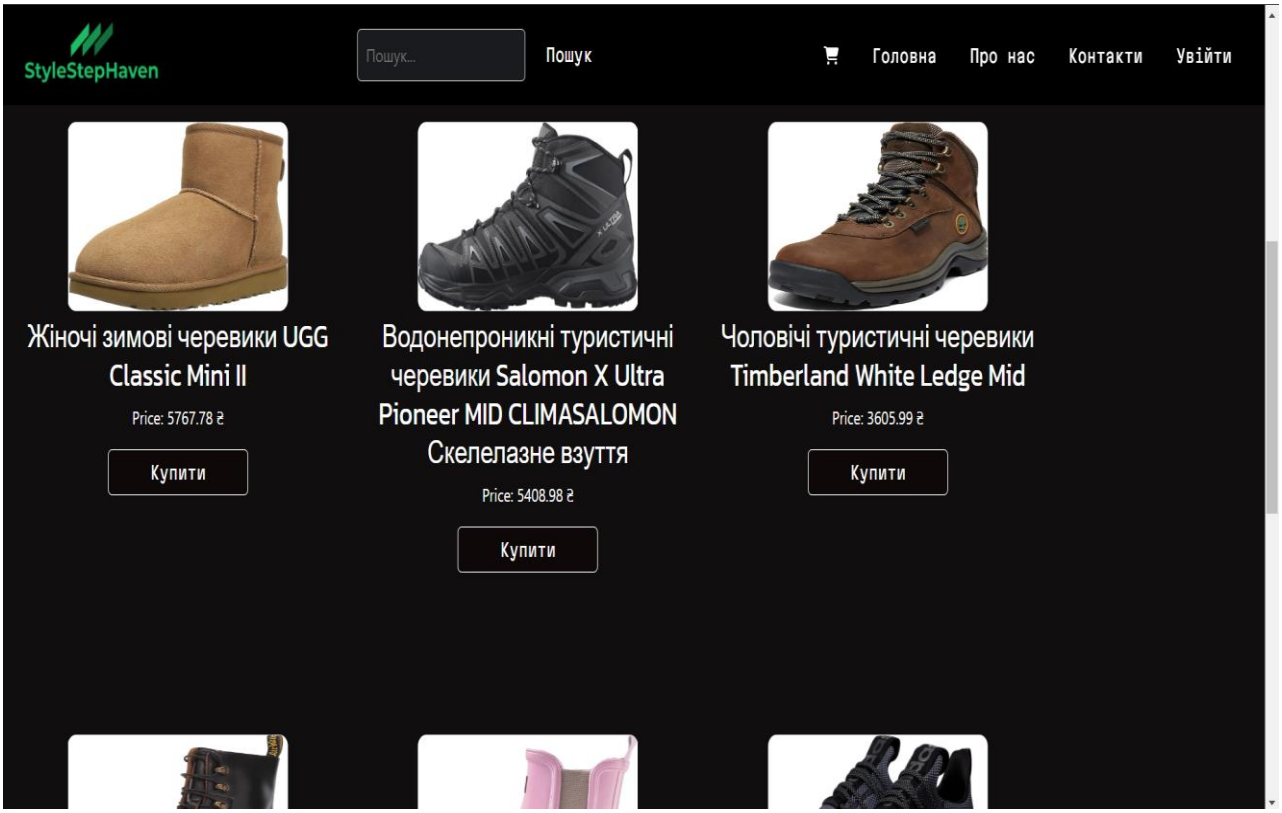

Рисунок 2.5 – Головна сторінка з товаром»

2.3 Висновки за розділом

Розробка основного алгоритму роботи інтернет-магазину є важливою складовою успішного функціонування платформи. У цьому розділі було детально розглянуто ключові компоненти алгоритму та його процеси. Зокрема, основний алгоритм роботи інтернет-магазину складається з таких елементів.

База даних товарів. Важлива для зберігання детальної інформації про всі товари, що доступні для продажу. Це забезпечує швидкий доступ до необхідних даних під час пошуку та фільтрації товарів.

Кошик покупця. Інтуїтивно зрозумілий інтерфейс для додавання, редагування та видалення товарів з кошика, що сприяє зручності користувачів при оформленні замовлення.

Система оплати. Інтеграція з різними платіжними шлюзами для безпечного та зручного проведення фінансових операцій, що підвищує довіру користувачів до магазину.

Завдяки використанню сучасних технологій і алгоритмів, інтернет-магазин здатен надавати високоякісні послуги, що задовольняють потреби користувачів, збільшують їхню зацікавленість та сприяють зростанню продажів. Таким чином, розробка і впровадження ефективного алгоритму роботи є ключем до успіху в конкурентному середовищі електронної комерції.

3 РОЗРОБКА ТА СТРУКТУРА ІНТЕРНЕТ МАГАЗИНУ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного засобу

Для розробки програмного додатку необхідно використати об’єктно-орієнтовану парадигму програмування. Дану парадигму підтримують такі мови як JavaScript та Python та їх фреймворки.

HTML (Hypertext Markup Language) – це стандартизована мова розмітки документів, яка використовується для перегляду веб-сторінок у браузері. Браузер отримує HTML-документ із сервера або відкриває його з локального диска за допомогою протоколу HTTP/HTTPS, а потім інтерпретує код в інтерфейс, який буде відображатися на екрані монітора [4].

CSS (каскадні таблиці стилів) – це спеціальна мова стилю сторінки [en], яка використовується для опису її зовнішнього вигляду [5].

Сама сторінка написана мовою розмітки даних. CSS, як HTML і JavaScript, є основною технологією Всесвітньої павутини. CSS найчастіше використовується для візуального рендерингу сторінок, написаних у HTML та XHTML, але форматування CSS також можна застосовувати до інших типів XML-документів.

JavaScript (JS) – це динамічна об’єктно-орієнтована мова програмування для створення прототипів. Впровадження стандарту ECMAScript. Найчастіше використовується для створення сценаріїв веб-сторінок, він надає клієнту (пристрою кінцевого користувача) можливість взаємодіяти з користувачем, контролювати браузер, асинхронно обмінюватися даними з сервером і змінювати структуру та зовнішній вигляд веб-сторінки [6].

JavaScript класифікується як прототип (підмножина об’єктно-орієнтованої) динамічно типізованої мови програмування сценаріїв. На додаток до прототипів, JavaScript також частково підтримує інші парадигми

програмування (імперативну та частково функціональну) і деякі пов’язані архітектурні властивості, включаючи: динамічну та слабку типізацію, автоматичне керування пам’яттю, успадкування прототипів, функції як об’єкти першого класу.

TypeScript – це мова програмування, запущена Microsoft у 2012 році, яка позиціонується як інструмент розробки веб-додатків, що розширює функціональні можливості JavaScript [7].

Розробником мови TypeScript є Андерс Хейлсберг, який раніше створив Turbo Pascal, Delphi і C#.

TypeScript сумісний із JavaScript і компілюється в нього. Насправді після компіляції додатки TypeScript можна запускати в будь-якому сучасному браузері або використовувати з серверною платформою Node.js.

TypeScript відрізняється від JavaScript тим, що він може явно статично виділяти типи, підтримує використання зрілих класів (таких як традиційні об’єктно-орієнтовані мови) і підтримує модулі підключення. Він призначений для збільшення швидкості розробки та полегшення читання, реконструкції та повторного використання коду Допомагає знайти помилки на етапах розробки та компіляції та може прискорити виконання програми.[7].

Redux – бібліотека для JavaScript з відкритим вихідним кодом, призначена для керування станом програми. Найчастіше використовують у зв'язку з React чи Angular розробки клієнтської части. Містить низку інструментів, що дозволяють значно спростити передачу даних сховища через контекст. Творці: Данило Абрамов та Ендрю Кларк [8].

Python – це інтерпретована об’єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією. Розроблено Гвідо ван Россумом у 1990 році. Високорівневі структури даних, а також динамічна семантика та динамічне зв’язування роблять його привабливим для швидкої розробки додатків і як засіб об’єднання існуючих компонентів. Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор Python і стандартна бібліотека доступні у

скомпільованому та вихідному коді на всіх основних платформах. Мова програмування Python підтримує кілька парадигм програмування, включаючи: об’єктно-орієнтоване, процедурне, аспектно-орієнтоване та функціональне[9].

React (стара назва: React.js, ReactJS) – це бібліотека JavaScript з відкритим вихідним кодом для створення користувальницьких інтерфейсів, призначена для вирішення проблеми часткового оновлення веб-вмісту, яка виникає під час розробки односторінкових програм. Розроблено Meta (раніше Facebook) і спільнотою окремих розробників [10].

React дозволяє розробникам створювати великомасштабні веб-програми, які можуть споживати дані, що змінюються з часом, без перезавантаження сторінки. Він прагне бути швидким, простим і масштабованим. React обробляє лише інтерфейс користувача в програмі.

Django – розширений відкритий фреймворк Python для розробки веб-систем. Його назвали на честь джазового музиканта Джанго Рейнхардта (на честь музичних уподобань одного із засновників проекту) [11].

Сайти Django складаються з однієї або кількох частин, і рекомендована модульність. Це одна зі значних архітектурних відмінностей між цим фреймворком та іншими фреймворками, такими як Ruby on Rails.

Архітектура Django схожа на Model-View-Controller (MVC). Однак те, що називається «контролерами» в класичній моделі MVC, у Django називається «представленнями», а те, що повинно бути «представленнями», називається «шаблонами». З цієї причини розробники Django називають MVC MTV ("Model-Template-View").

Проаналізувавши мови програмування, було вирішено використовувати мову програмування Python. Для розробки веб-інфтерфейсу вирішено використати JavaScript.

3.2 MVC – архітектурний шаблон додатку

Основною задачею інтернет-магазину є надання інформації про доступний товар та можливість придбати його. Для представлення даних було обрано концепцію MVC (model - view – controller) [12].

Контролер (controller) представляє клас, що забезпечує зв'язок між користувачем та системою, представленням та сховищем даних. Він отримує дані, що вводяться користувачем, і обробляє їх. І в залежності від результатів обробки надсилає користувачеві певний висновок, наприклад, у вигляді представлення. Представлення (view) – це власне візуальна частина або інтерфейс програми користувача. Як правило, це html-сторінка, яку користувач бачить, зайшовши на сайт.

Модель (Model) відповідає за обробку даних та їх збереження в базі даних або іншому сховищі. Вона повинна забезпечувати доступ до даних, перевірку їх цілісності та валідацію. Модель може включати додаткову логіку для оптимізації запитів до бази даних, перевірки прав доступу та реалізації взаємодії зіншими компонентами.

Така архітектура є чудовим рішенням для розробки веб-застосунків, оскількизабезпечує розподіл логіки між компонентами та є зрозумілою для більшості черезїї популярність (рис. 3.1).

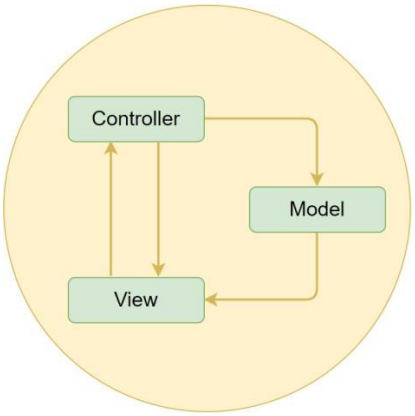

Рисунок 3.1 – Архітектурний шаблон MVC

У Django MTV є широко використовуваною архітектурою для розробки

веб-додатків. Рівень моделі відповідає за зберігання та обробку даних у базі даних або іншому сховищі даних. Вона залишається незмінною порівняно з архітектурою MVC (рис. 3.2). Рівні перегляду в Django вважаються "шаблоном представлень". Шаблон відповідає за відображення даних і форматування їх у зрозумілу форму, а представлення відповідає за обробку запитів користувачів і передачу даних у шаблон [11].

Це дозволяє розподілити логіку між цими двома компонентами та дає змогу точніше реалізувати рівень перегляду порівняно з архітектурою MVC. Рівень презентації є ключовим компонентом в архітектурі багатьох сучасних програмних систем, що відповідає за відображення даних у зручному та зрозумілому форматі.

Цей рівень діє як посередник між користувачем і системою, перетворюючи технічні деталі та інформацію в інтуїтивно зрозумілий і візуально привабливий формат. Усі елементи інтерфейсу користувача, такі як кнопки, форми введення, списки, таблиці, графіки та інші віджети, розташовані на цьому рівні. Крім того, рівень презентації може включати додаткову логіку, яка спрощує процес відображення даних. Така логіка може включати, наприклад, визначення порядку відображення елементів, умов для відображення певних елементів на основі статусу системи або користувача та форматування інформації відповідно до певних стандартів або вимог користувача.

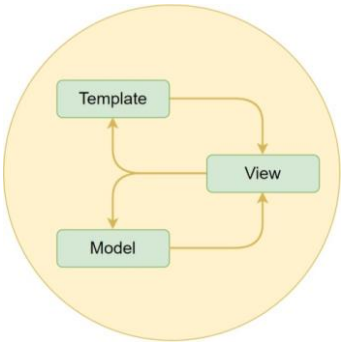

Рисунок 3.2 – Архітектурний шаблон MVT

Завдяки простій і практичній архітектурі фреймворк Django має такі переваги перед іншими фреймворками [11].

Швидкість розробки: фреймворки спрощують процес створення додатків і дозволяють розробникам швидко перетворювати ідеї на готові проекти. Економите час і ресурси на кожному етапі від ідеї до розгортання.

Повнофункціональний: Django надає доступ до великої кількості функцій, які значно полегшують розробку.

Серед них зручна система авторизації та автентифікації, функції, які можна змінювати за потреби, і налаштування відображення бази даних в панелі адміністратора.

Безпека: Django забезпечує високий рівень безпеки для захисту вашого проекту від потенційних загроз.

Ефективно використовуйте механізми входу та пароля та системи автентифікації користувачів, щоб уникнути помилок.

Масштабованість: Django має великий потенціал для використання у великих проектах із великим обсягом трафіку.

Багато веб-сайтів із високим трафіком обирають Django для обробки великих запитів.

Універсальність: Можливість створювати власні розширення, які можна підключити до вашого проекту, написавши лише кілька рядків або використовуючи інші розширення, не змінюючи програмний код проекту.

3.3 Фреймворк Django для розробки веб-додатків

Django REST Framework (DRF) – потужний інструмент для розробки веб-сервісів на основі Django Framework. DRF надає простий і зручний спосіб створення та розгортання API для взаємодії з клієнтами на різних платформах і пристроях.Однією з головних переваг DRF є те, що він заснований на фреймворку Django, який має багатофункціональну архітектуру та підходить для розробки високоякісних веб-додатків [13].

Крім того, DRF надає кілька вбудованих інструментів, які дозволяють швидко розробляти API. Ще однією важливою перевагою DRF є можливість створювати API у різних форматах, таких як JSON, XML і YAML. Крім того,

DRF надає різні API-інтерфейси автентифікації, авторизації і безпеки, такі як Django-OAuth Toolkit для підтримки автентифікації OAuth2.

Однією з головних переваг є швидкість розробки. Django REST framework надає багато готових рішень, які значно спрощують створення API. Наприклад, вінмає вбудований підтримку автентифікації, серіалізації даних, валідації форм, підтримки різних форматів даних тощо (рис. 3.3). Також він має велику кількість документації та готових прикладів, що дозволяє швидко засвоїти фреймворк.

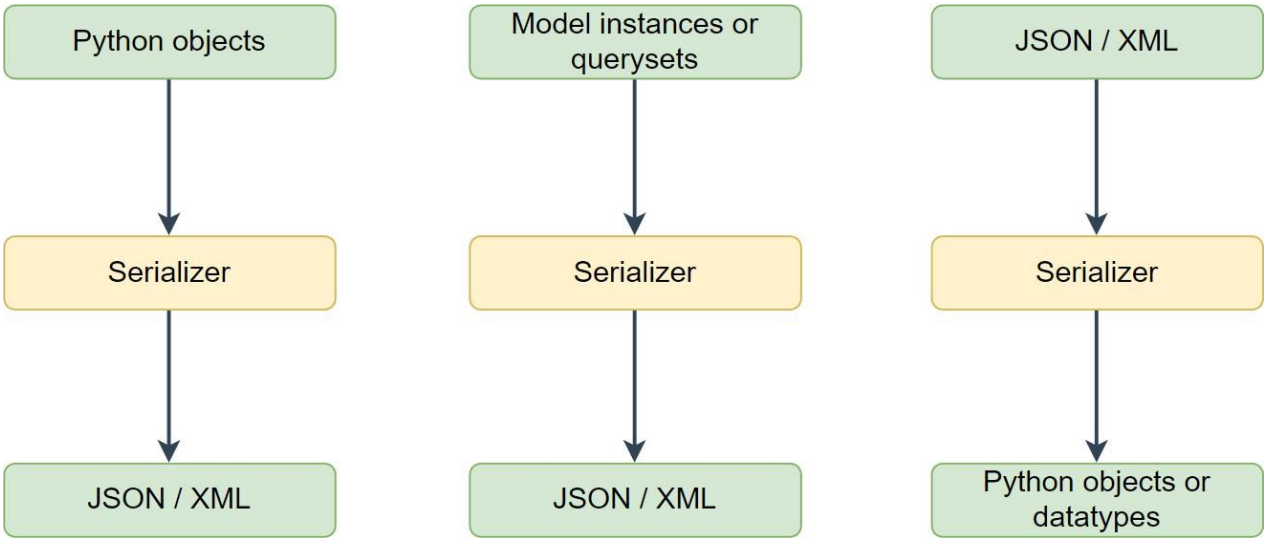

Рисунок 3.3 – Схема конвертації в різні формати за допомогою Serializer

DRF надає зручні функції для написання тестів для перевірки функціональності API. Це дозволяє швидко виявити і виправити можливі помилки. Однією з головних переваг DRF є простота використання. Цей фреймворк дозволяє легко створювати моделі, перегляди та URL-адреси API.Фреймворк Django REST базується на архітектурі Model-View-Controller (MVC), яка дозволяє розробникам використовувати той самий підхід до написання, що й основний фреймворк Django, що полегшує розробку, тестування та підтримку API процес спрощується.Крім того, він надає готовий

набір різних методів HTTP-запитів, які можна використовувати, щоб переконатися, що ваші запити та відповіді обробляються правильно [14].

Використання тестування є важливою складовою розробки програмного забезпечення. Django REST framework дозволяє тестувати API як через браузер, такі через unit тести (рис. 3.4). Обидва методи дозволяють перевірити систему на помилки та недоліки в API та забезпечити гарантію відсутності відхилень у роботіготової системи.

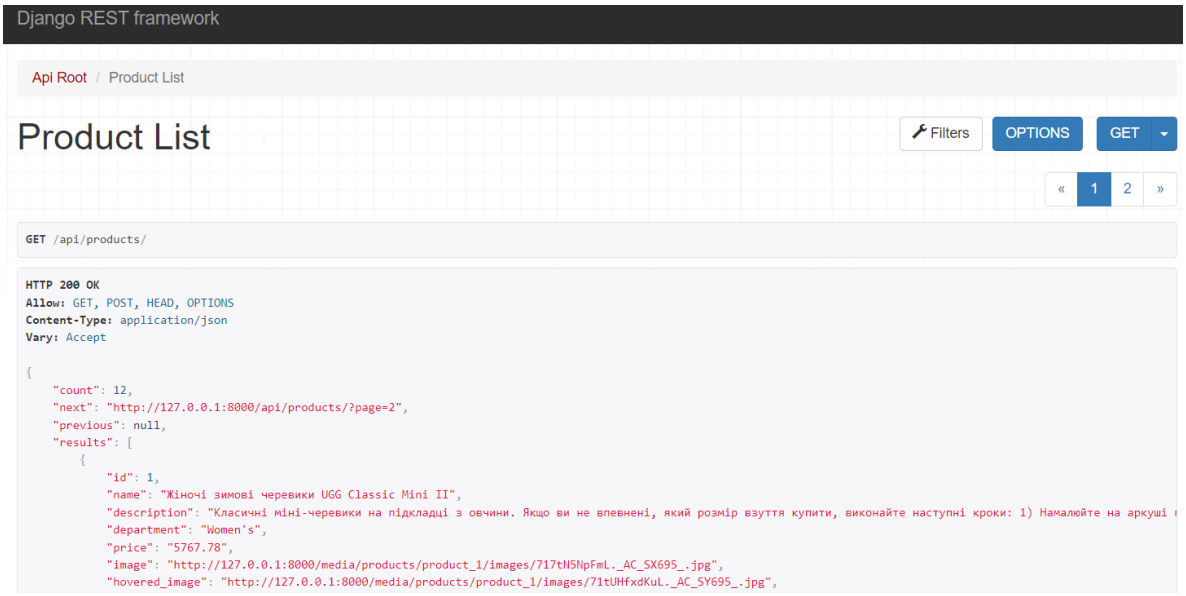

Рисунок 3.4 – Веб сторінка DRF

Тестування веб-переглядача часто використовується для перевірки API у веб-середовищі та може тестувати різні типи запитів, наприклад GET, POST, PUT, DELETE та PATCH. Цей метод тестування полегшує перевірку функціональності API з точки зору кінцевого користувача. Платформа Django REST має вбудовану підтримку кросбраузерного тестування, тому ви можете використовувати тести Django для інтеграції з тестуванням REST API.

Модульне тестування є більш ефективним методом тестування, оскільки воно дозволяє тестувати окремі компоненти вашої системи та перевіряти, як поводиться ваш код, не запускаючи всю програму.

Модульні тести дозволяють перевірити, чи правильно функціонують функції, методи та класи незалежно від загальної системи. Фреймворк Django REST забезпечує вбудовану підтримку модульного тестування. Для тестування використовуються такі фреймворки, як Pytest, Unittest і Django Test Framework. Ці фреймворки дозволяють вам писати тести, які перевіряють функціональність кожного елемента вашого API окремо [15].

Тому платформа Django REST є потужним, гнучким і практичним інструментом для розробки REST API. Він надає розробникам широкий спектр функцій, які спрощують процес розробки та підтримки API і забезпечують високоякісну кодову базу.

3.4 Фреймворк React для розробки веб-інтерфейсу

React – це бібліотека JavaScript для створення користувацьких інтерфейсів, розроблена компанією Facebook. Вона використовується для побудови веб-додатків з високою взаємодією та динамічною зміною вмісту сторінок [10].

Основні особливості React.

Компонентний підхід: React базується на концепції компонентів. Кожен компонент - це незалежна одиниця, яка має свій власний стан та властивості. Це сприяє модульності, повторному використанню та покращує читабельність коду.

Virtual DOM: React використовує віртуальний DOM для оптимізації взаємодії з реальним DOM. Він зберігає копію структури DOM у пам'яті та оновлює її лише у випадку зміни стану компонентів. Це дозволяє зменшити кількість маніпуляцій з реальним DOM і підвищує продуктивність додатків.

Життєвий цикл компонентів: кожен React компонент має свій власний життєвий цикл, який складається з різних етапів, таких як створення, оновлення та видалення. Це дозволяє реагувати на різні події та здійснювати необхідні дії.

Односторінкові додатки (SPA): React дозволяє легко створювати односторінкові додатки, які завантажуються один раз, а далі взаємодіють з користувачем без перезавантаження сторінок.

Декларативний підхід: React пропонує декларативний спосіб опису візуального відображення, що спрощує роботу з UI та забезпечує більшу прозорість коду.

Розширюваність: React дозволяє легко інтегрувати різні бібліотеки та розширювати функціонал за допомогою додаткових модулів.

React широко використовується у веб-розробці та є однією з найпопулярніших технологій для створення користувацьких інтерфейсів. Він підтримується активною спільнотою розробників та має велику кількість розширень та інструментів для розробки.

3.5 Основна структура клієнтської частини додатку

Основна структура клієнтської частини додатку зображена на рис. 3.5.

```
src > App.tsx > ...
22  function App() {
34
35      return (
36          <div className={styles.App}>
37              <BrowserRouter>
38                  {/* Відображаємо HeaderAuth, якщо немає access token, та Header, якщо є access token */}
39                  {hasAccessToken ? <HeaderAuth isLoggedIn={isLoggedIn} /> : <Header isLoggedIn={isLoggedIn} />}
40                  <Routes>
41                      <Route path="basket" element={<Basket />} />
42                      <Route path="/" element={<Home />} />
43                      <Route path="about" element={<About />} />
44                      <Route path="contacts" element={<Contacts />} />
45                      <Route path="login" element={<Login setIsLoggedIn={setIsLoggedIn} />} />
46                      <Route path="profile" element={<Profile isLoggedIn={isLoggedIn} />} />
47                      <Route path="registration" element={<Registration />} />
48                      <Route path="singleProduct/:id" element={<SingleProduct />} />
49                      <Route path="searchPage" element={<SearchPage />} />
50                      <Route path="men" element={<Men />} />
51                      <Route path="women" element={<Women />} />
52                      <Route path="kids" element={<Kids />} />
53                      <Route path="*" element={<NotFound />} />
54                  </Routes>
55                  <Footer />
56              </BrowserRouter>
57          </div>
58      );
```

Рисунок 3.5 – Структура клієнтської частини

Home – головна сторінка веб-додатку.

Basket – кошик для товарів, які додав користувач.

About – сторінка з інформацією про інтернет-магазин.

Contacts – сторінка з інформацією для зв’язку.

Login – сторінка авторизації.

Registration – сторінка реєстрації.

Profile – сторінка профілю з інформацією про користувача.

SingleProduct – сторінка конкретного товару з інформацією про нього.

SearchPage – сторінка з результатами пошуку.

Men – сторінка категорії товарів «Чоловіки».

Women – сторінка категорії товарів «Жінки».

Kids – сторінка категорії товарів «Діти».

HeaderAuth – хидер авторизованого користувача.

Header – хидер неавторизованого користувача.

Footer – футер сторінки.

Огляд компоненту Home (рис. 3.6).

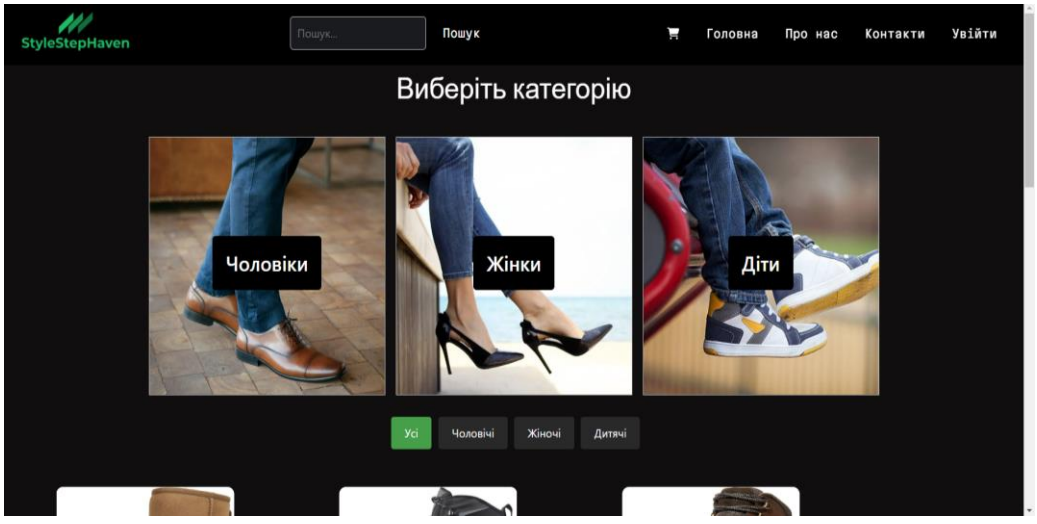

Рисунок 3.6 – Компонент Home

На сторінці є кілька ключових компонентів, які забезпечують зручну навігацію та комфортний процес вибору і придбання товарів.

Header: цей компонент знаходиться у верхній частині сторінки і відіграє важливу роль у забезпеченні доступу до основних функцій сайту. В Header розміщений кошик, куди додаються обрані товари. Крім того, тут можуть бути присутні логотип компанії, меню навігації та пошуковий рядок для швидкого пошуку товарів [16].

Footer: розташований у нижній частині сторінки, цей компонент містить додаткову інформацію, таку як контактні дані, посилання на соціальні мережі, інформація про компанію, умови використання сайту та політику конфіденційності. Footer допомагає користувачам швидко знайти важливу інформацію і зв’язатися з підтримкою у разі потреби.

ShoeCategories: цей компонент дозволяє користувачам обирати категорію потрібних товарів, наприклад, спортивне взуття, класичне взуття, зимове взуття тощо. Кожна категорія представляє собою набір товарів, що відповідають певним критеріям, що полегшує процес пошуку і вибору.

Filter: компонент фільтрації надає можливість звузити вибір товарів за різними параметрами, такими як ціна, бренд, розмір, колір і матеріал. Це значно спрощує процес пошуку конкретного товару, що відповідає індивідуальним потребам користувача.

Product: кожен товар представлений у вигляді окремого компонента, де детально описані його характеристики, ціна, наявність і можливість додати його в кошик. Цей компонент також може містити фотографії товару, відгуки користувачів та іншу корисну інформацію. На цій сторінці користувач може легко вибрати потрібну категорію товарів за допомогою компонента ShoeCategories, відфільтрувати їх за допомогою Filter та переглянути детальну інформацію про кожен продукт у компоненті Product. Додавши обрані товари до кошика, який знаходиться в Header, користувач може перейти до оформлення замовлення, що робить процес купівлі швидким та зручним [17].

3.6 Огляд компоненту Basket

На скріншоті зображено компонент Basket (рис. 3.7).

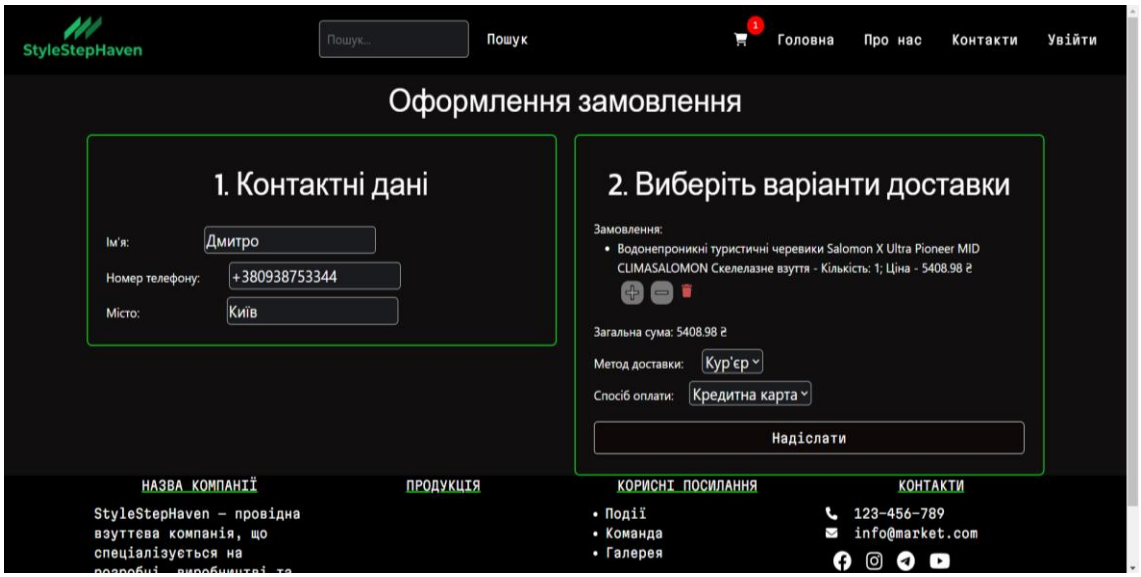

Рисунок 3.7 – Компонент Basket

На цій сторінці є кілька важливих компонентів, таких як ContactForm та DeliveryForm. Давайте детальніше розглянемо кожен з них.

ContactForm: цей компонент використовується для введення контактних даних користувача.

Функціональність: користувач може ввести своє ім'я, номер телефону та місто. Ця інформація необхідна для обробки замовлення та зв'язку з користувачем у разі необхідності.

DeliveryForm: цей компонент відповідає за управління деталями замовлення.

Функціональність: Перегляд замовлення: Користувач може переглянути поточні товари у своєму кошику, включаючи їх кількість і загальну вартість.

Зміна кількості: користувач може збільшувати або зменшувати кількість кожного товару в кошику, що автоматично оновлює загальну вартість замовлення [18].

Метод доставки: Користувач може вибрати бажаний метод доставки.

Спосіб оплати: користувач може вибрати зручний для нього спосіб оплати.

Процес Замовлення (рис. 3.8).

Після введення всіх необхідних даних у формах ContactForm та DeliveryForm, користувач може натиснути кнопку «Надіслати». Далі відбувається наступне.

Підтвердження та перевірка. Після натискання кнопки «Надіслати» усі введені дані перевіряються на коректність. Якщо всі поля заповнені правильно, замовлення підтверджується і користувача перенаправляють на наступний етап.

Перехід на сторінку оплати. Користувача автоматично перенаправляють на сторінку оплати через платіжну систему Fondy. Fondy - це популярна платіжна платформа, яка забезпечує безпечні та зручні фінансові транзакції онлайн.

Введення «Даних Про Картку». На сторінці оплати Fondy користувач повинен ввести дані своєї банківської картки. Ці дані включають номер картки, термін її дії та CVV код. Fondy забезпечує надійний захист цих даних, використовуючи сучасні методи шифрування та безпеки.

Підтвердження «Оплати». Після введення всіх необхідних даних про картку користувач підтверджує оплату. Система Fondy обробляє транзакцію і, в разі успішного завершення, відображає повідомлення про успішну оплату. Якщо оплата пройшла успішно, користувач отримує підтвердження на екрані, а також електронною поштою, якщо це було передбачено налаштуваннями.

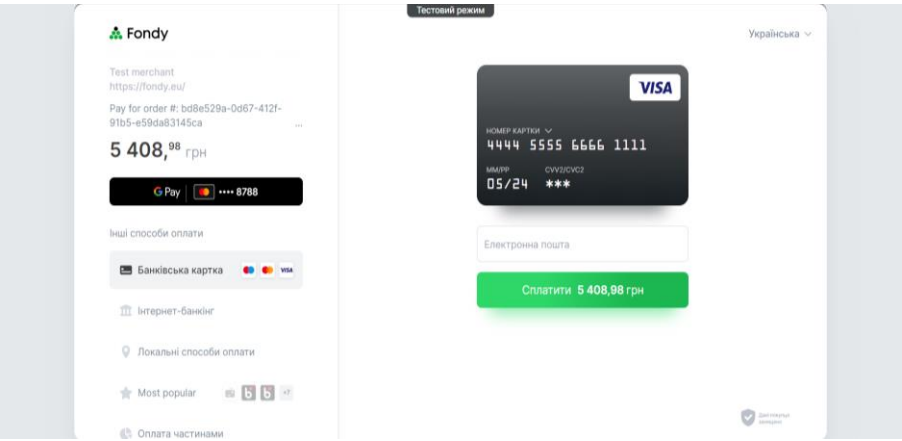

Рисунок 3.8 – Сторінка оплати замовлення

3.7 Огляд компоненту About

На скріншоті зображено сторінку компоненту About (рис. 3.9 та рис. 3.10).

Рисунок 3.9 – Компонент About

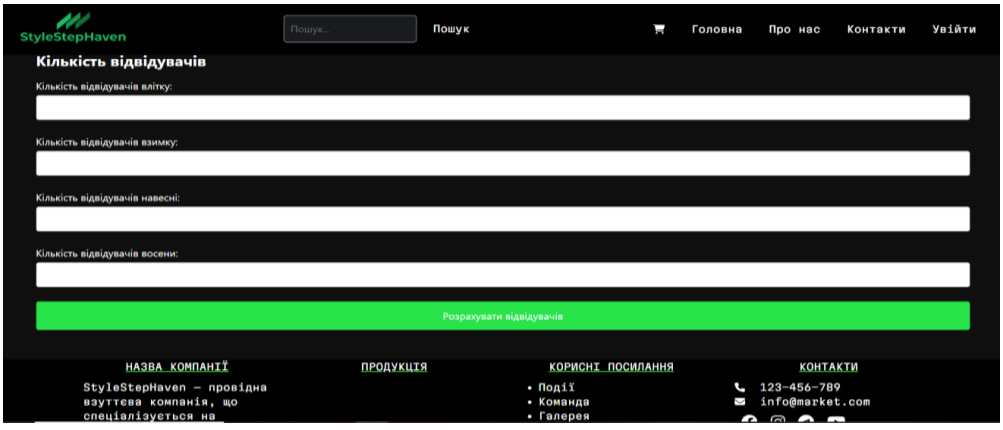

Рисунок 3.10 – Компонент About

На рис. 3.9 представлена інформація про інтернет-магазин, яка детально розкриває його історію, місію та команду. Тут ви можете дізнатися про шлях розвитку магазину від моменту його заснування до сьогоднішнього дня, ознайомитися з основними цінностями та цілями, які він ставить перед собою.

На рис. 3.10 показана математична модель, яка прогнозує кількість відвідувачів інтернет-магазину залежно від сезону. Ця модель враховує сезонні коливання та інші фактори, що впливають на відвідуваність, дозволяючи точно розрахувати і планувати ресурси магазину в різні періоди року.

3.8 Огляд компоненту Contacts

На скріншоті зображено сторінку компоненту Contacts (рис. 3.11).

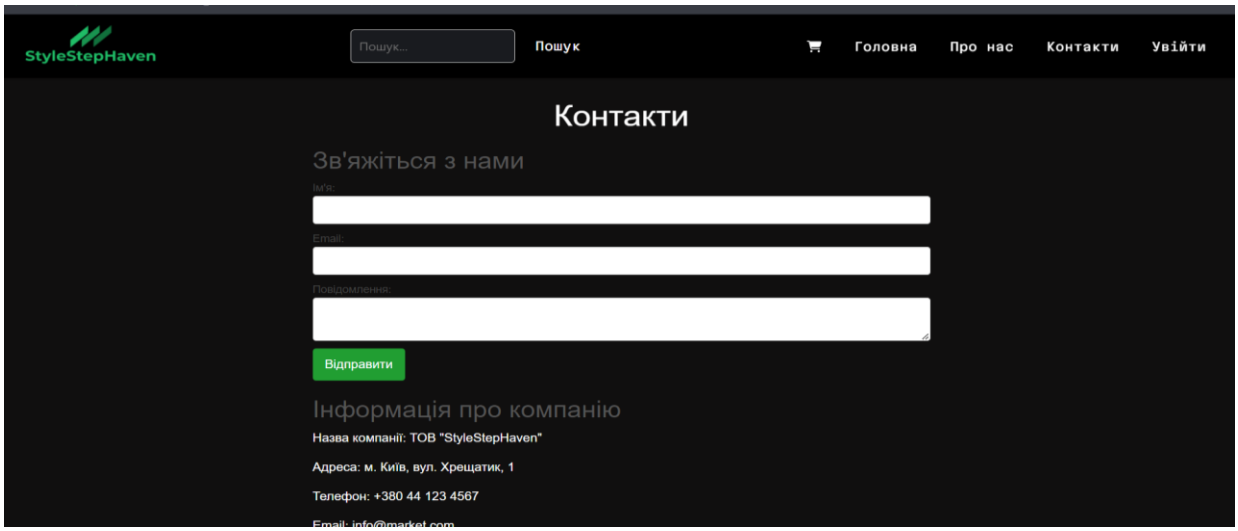

Рисунок 3.11 – Компонент Contacts

На рис. 3.11 в компоненті Contacts представлена вичерпна інформація для зручного зв'язку з інтернет-магазином. Тут ви знайдете такі дані:

- Контактні дані: включають номер телефону, електронну адресу та посилання на соціальні мережі, що дозволяють швидко зв'язатися з представниками компанії для отримання консультацій або вирішення будь-яких питань.

- Розташування компанії: тут наведена точна адреса офісу або складу компанії, а також інтерактивна карта, що дозволяє легко знайти шлях до компанії.

Ця інформація надає користувачам повне уявлення про те, як зв'язатися з компанією та де знаходиться її фізичне розташування.

3.9 Огляд компонентів Registration та Login

На скріншоті зображено сторінку компонентів Registration та Login (рис. 3.12 та рис. 3.13).

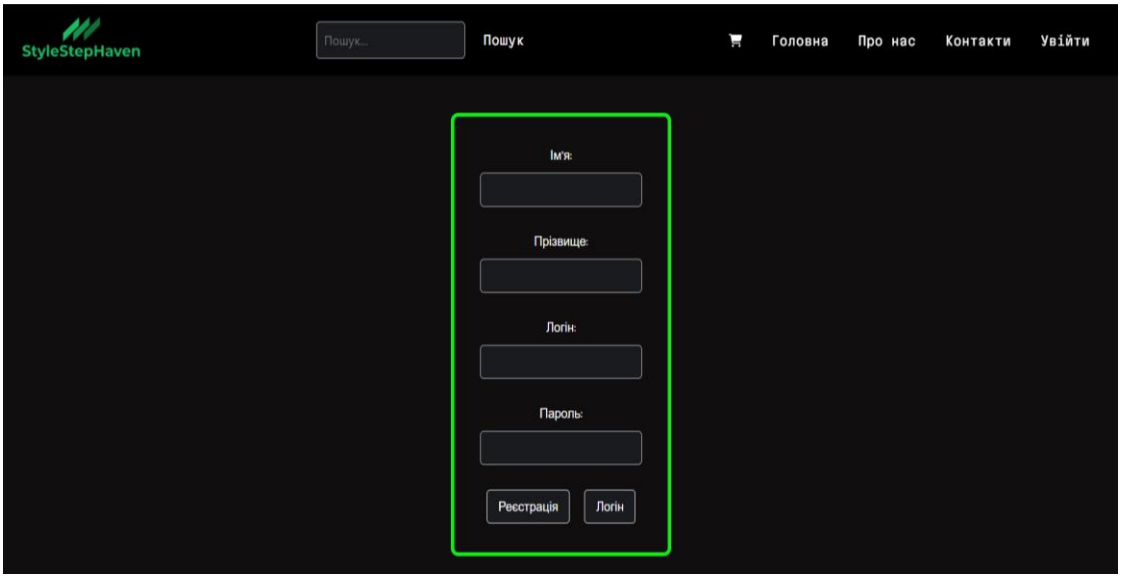

Рисунок 3.12 – Компонент Registration

На рис. 3.12 зображено вікно реєстрації, яке дозволяє новим користувачам створити обліковий запис в інтернет-магазині. У цьому вікні необхідно ввести наступні дані:

- Ім'я: поле для введення вашого імені, яке буде використовуватись для персоналізації вашого облікового запису та спілкування з вами.

- Прізвище: поле для введення вашого прізвища, що також може використовуватися для ідентифікації та персоналізації.

- Логін: унікальне ім'я користувача, яке буде використовуватися для входу в систему. Це ім'я повинно бути унікальним і легко запам'ятовуватися.

- Пароль: поле для створення пароля, який забезпечує безпеку вашого облікового запису. Рекомендується використовувати складний пароль, що містить комбінацію літер, цифр та спеціальних символів.

На рис. 3.13 зображена форма авторизації (логіну), яка використовується для входу в обліковий запис існуючих користувачів. У цій формі потрібно ввести наступні дані:

- Ім'я користувача: поле для введення вашого логіна, який був створений під час реєстрації. Це ім'я ідентифікує вас як зареєстрованого користувача.

- Пароль: поле для введення вашого пароля. Він забезпечує доступ до вашого облікового запису і захищає вашу персональну інформацію.

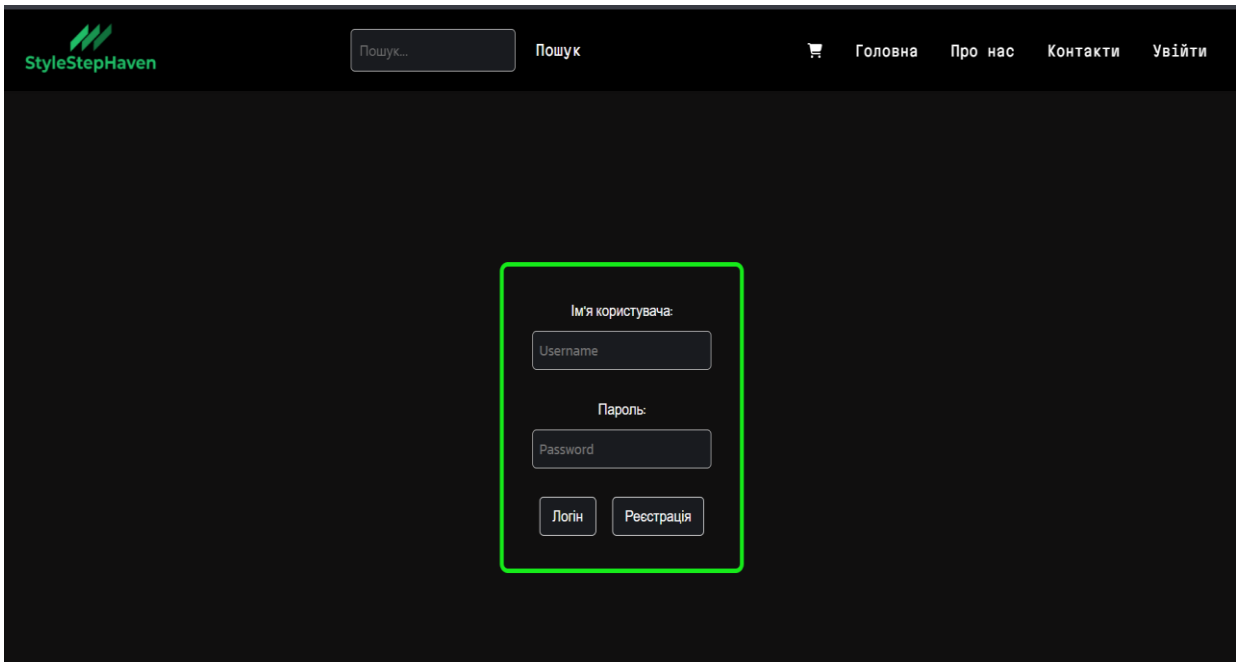

Рисунок 3.13 – Компонент Login

Ці форми забезпечують процес реєстрації нових користувачів та авторизації існуючих, що є необхідними кроками для забезпечення безпеки та персоналізованого доступу до функціоналу інтернет-магазину.

3.10 Висновки за розділом

Під час аналізу варіантів мов програмування та фреймворків для розробки програмного додатку було вирішено використовувати Python для серверної частини та JavaScript для клієнтської частини. Python, зокрема у поєднанні з фреймворком Django REST, забезпечує потужну підтримку для створення REST API, пропонуючи вбудовані можливості для модульного та кросбраузерного тестування. Це дозволяє розробникам ефективно перевіряти функціональність API з точки зору кінцевого користувача та забезпечувати якісну кодову базу.

JavaScript, зокрема за допомогою бібліотеки React, дозволяє створювати динамічні та інтерактивні веб-інтерфейси, що є ключовим для забезпечення зручності користувацького досвіду. Використання TypeScript, який розширює можливості JavaScript, дозволяє покращити процес розробки завдяки явній статичній типізації та підтримці модулів, що сприяє швидшій розробці та легшому підтриманню коду [19].

Таким чином, обраний стек технологій, що включає Python з Django REST для серверної частини та JavaScript з React для клієнтської частини, є оптимальним для створення сучасного, масштабованого та ефективного веб-додат

У даному розділі була розглянута основна структура клієнтської частини веб-додатку інтернет-магазину. Було детально описано кожний компонент, що складає інтерфейс користувача. Клієнтська частина включає такі основні сторінки, як головна сторінка (Home), кошик для товарів (Basket), інформаційні сторінки (About, Contacts), а також сторінки для роботи з обліковим записом користувача (Login, Registration, Profile) [20].

Компоненти HeaderAuth та Header були розділені для забезпечення різного відображення інтерфейсу для авторизованих та неавторизованих користувачів, що сприяє більш персоналізованому досвіду взаємодії з додатком. Footer ж забезпечує єдине оформлення нижньої частини сторінки для всіх розділів сайту.

4 РОЗРОБКА МАТЕМАТИЧНИХ МОДЕЛЕЙ

4.1 Математична модель кількості відвідувачів у різні сезони року

Математична модель кількості відвідувачів у різні сезони року допомагає зрозуміти та передбачити зміни у відвідуваності об'єкта або заходу залежно від сезону.

Опис коду.

1. Створення станів для зберігання даних про відвідувачів у різні сезони:

```
const [summerVisitors, setSummerVisitors] = useState<number | "">("");
const [winterVisitors, setWinterVisitors] = useState<number | "">("");
const [springVisitors, setSpringVisitors] = useState<number | "">("");
const [fallVisitors, setFallVisitors] = useState<number | "">("");
```

Ці рядки коду створюють стани (використовуючи хук `useState`) для зберігання кількості відвідувачів для кожного сезону: літо, зима, весна та осінь. Спочатку всі значення станів порожні (""), але вони можуть бути встановлені як число.

2. Функція для обчислення відвідувачів на основі кількості літніх відвідувачів:

```
const calculateFromSummer = (summer: number) => {
  setWinterVisitors(summer * 0.6);
  setSpringVisitors(summer * 0.8);
  setFallVisitors(summer * 0.8);
};
```

Ця функція приймає кількість літніх відвідувачів і обчислює кількість відвідувачів для інших сезонів за формулами:

- Зима = Літо * 0.6

- Весна = Літо * 0.8

- Осінь = Літо * 0.8

3. Функція для обчислення відвідувачів на основі кількості зимових відвідувачів:

```
const calculateFromWinter = (winter: number) => {
  const summer = winter * (10 / 6);
  calculateFromSummer(summer);
  setSummerVisitors(summer);
};
```

Ця функція приймає кількість зимових відвідувачів і обчислює кількість літніх відвідувачів за формулою:

- Літо = Зима * (10 / 6)

Потім викликає функцію `calculateFromSummer` для обчислення відвідувачів для інших сезонів на основі обчислених літніх відвідувачів.

4. Функція для обчислення відвідувачів на основі кількості весняних відвідувачів:

```
const calculateFromSpring = (spring: number) => {
  const summer = spring * 1.25;
  calculateFromSummer(summer);
  setSummerVisitors(summer);
};
```

Ця функція приймає кількість весняних відвідувачів і обчислює кількість літніх відвідувачів за формулою:

- Літо = Весна * 1.25

Потім викликає функцію `calculateFromSummer` для обчислення відвідувачів для інших сезонів на основі обчислених літніх відвідувачів.

5. Функція для обчислення відвідувачів на основі кількості осінніх відвідувачів:

```
const calculateFromFall = (fall: number) => {
  const summer = fall * 1.25;
  calculateFromSummer(summer);
  setSummerVisitors(summer);
};
```

Ця функція приймає кількість осінніх відвідувачів і обчислює кількість літніх відвідувачів за формулою:

- Літо = Осінь * 1.25

Потім викликає функцію `calculateFromSummer` для обчислення відвідувачів для інших сезонів на основі обчислених літніх відвідувачів.

6. Функція для обробки обчислення відвідувачів:

```
const handleCalculate = () => {
  if (typeof summerVisitors === "number" && summerVisitors > 0) {
    calculateFromSummer(summerVisitors);
  } else if (typeof winterVisitors === "number" && winterVisitors > 0) {
    calculateFromWinter(winterVisitors);
  } else if (typeof springVisitors === "number" && springVisitors > 0) {
    calculateFromSpring(springVisitors);
  } else if (typeof fallVisitors === "number" && fallVisitors > 0) {
    calculateFromFall(fallVisitors);
```

```
        } else {
            alert("Будь ласка, введіть дійсне число відвідувачів для одного із сезонів");
        }
    };
```

Ця функція перевіряє, яке значення для відвідувачів (літо, зима, весна або осінь) введено і є дійсним числом. Залежно від цього значення викликається відповідна функція для обчислення кількості відвідувачів для всіх сезонів. Якщо жодне значення не введено або воно недійсне, виводиться повідомлення про помилку.

Скріншоти роботи програми (рис 4.1 та рис. 4.2).

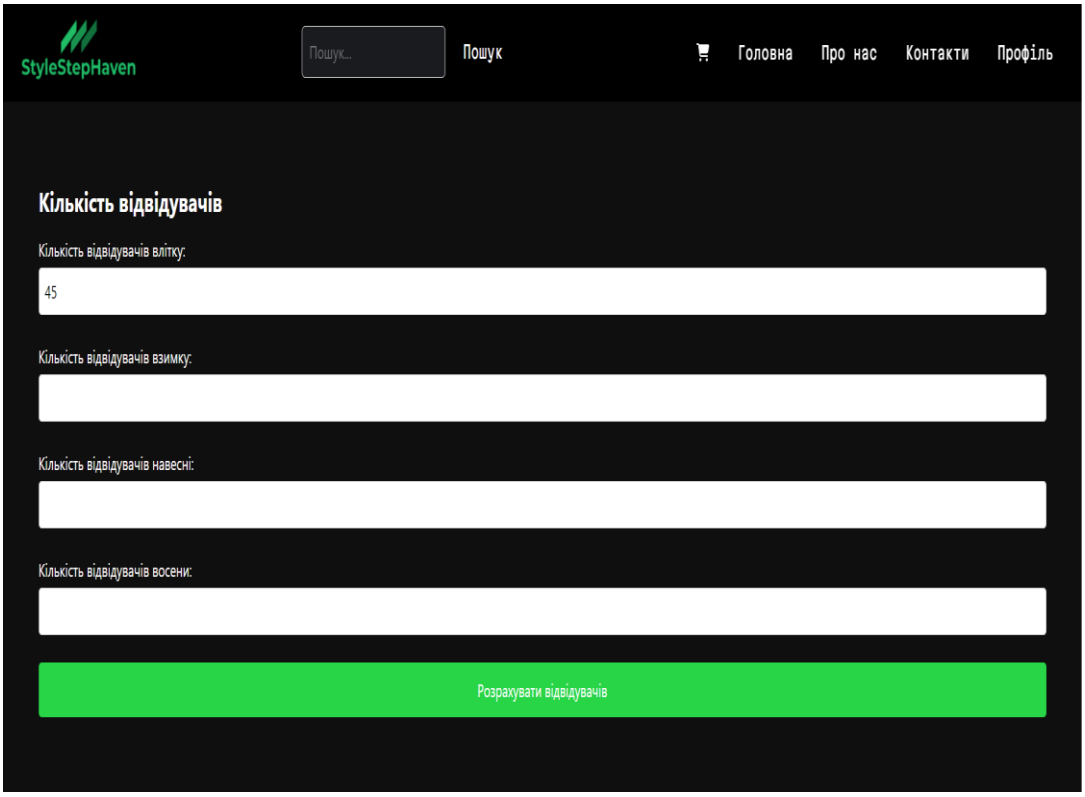

Рисунок 4.1 – Вводимо дані про відвідувачів влітку та натиснемо кнопку «Розрахувати відвідувачів»

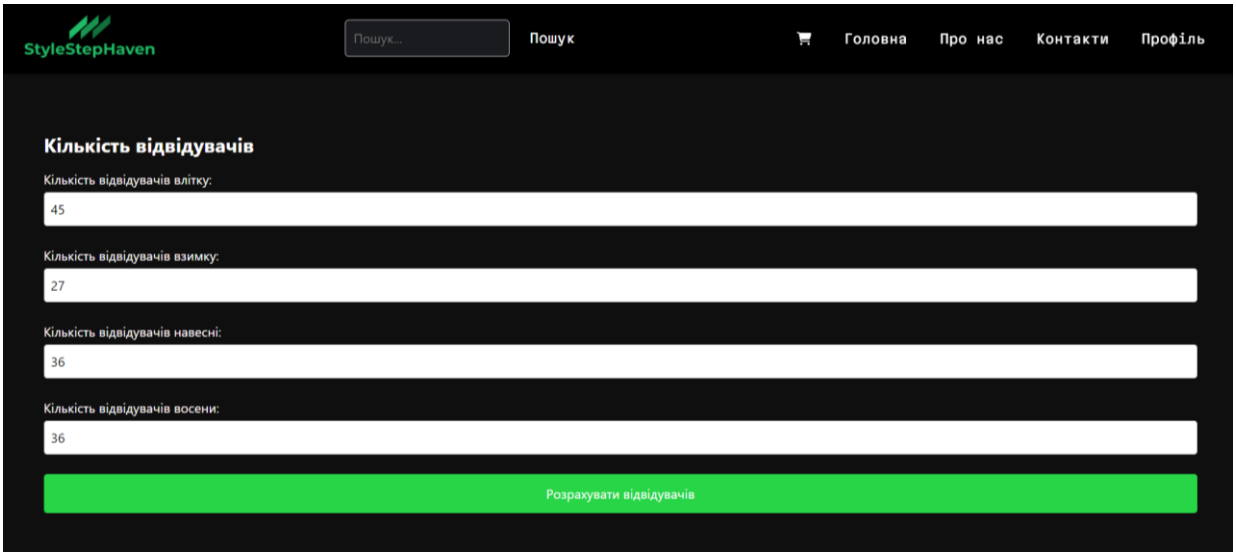

Рисунок 4.2 – Результати розрахунків

4.2 Математична модель «трафік і поведінка користувачів»

Трафік і поведінка користувачів:

Потік користувачів (U): кількість унікальних відвідувачів за певний період, U(t).

Час перебування на сайті (T): Середній час, який користувач проводить на сайті.

$$T = \frac{\sum_{i=1}^{N} t_i}{N}, \tag{4.1}$$

де t_i – час перебування кожного з N користувачів на сайті.

Показник відмов (B): Відсоток користувачів, які залишають сайт після перегляду однієї сторінки.

$$B = \frac{U_{single\ page}}{U_{total}} \times 100\%, \tag{4.2}$$

де $U_{\text{single_page}}$ – кількість відвідувачів, які переглянули лише одну сторінку,

U_{total} — загальна кількість відвідувачів.

Реалізація у React компоненті

1. Потік користувачів (U):

- Зберігається у стані `uniqueVisitors`.

- Задається через форму введення:

const [uniqueVisitors, setUniqueVisitors] = useState<number | "">("");

<input type="number" value={uniqueVisitors} onChange={(e) => setUniqueVisitors(Number(e.target.value) || "")} />

2. Час перебування на сайті (T):

- Середній час перебування зберігається у стані `averageTimeOnSite`.

- Задається через форму введення:

const [averageTimeOnSite, setAverageTimeOnSite] = useState<number | "">("");

<input type="number" value={averageTimeOnSite} onChange={(e) => setAverageTimeOnSite(Number(e.target.value) || "")} />

3. Показник відмов (B):

- Вводяться значення `singlePageVisits` та `totalVisits`.

- Вони зберігаються у відповідних станах:

const [singlePageVisits, setSinglePageVisits] = useState<number | "">("");

const [totalVisits, setTotalVisits] = useState<number | "">("");

- Формула для розрахунку показника відмов використовується у функції:

```
const calculateBounceRate = (singlePageVisits: number, totalVisits: number) => {
  if (totalVisits === 0) return 0;
  return (singlePageVisits / totalVisits) * 100;
};
const handleTrafficCalculate = () => {
  if (typeof singlePageVisits === "number" && typeof totalVisits === "number") {
    setBounceRate(calculateBounceRate(singlePageVisits, totalVisits));
  } else {
    alert("Будь ласка, введіть дійсні числа для відвідувань однієї сторінки та загальних відвідувань");
```

Скріншоти роботи програми (рис. 4.3 та рис. 4.4).

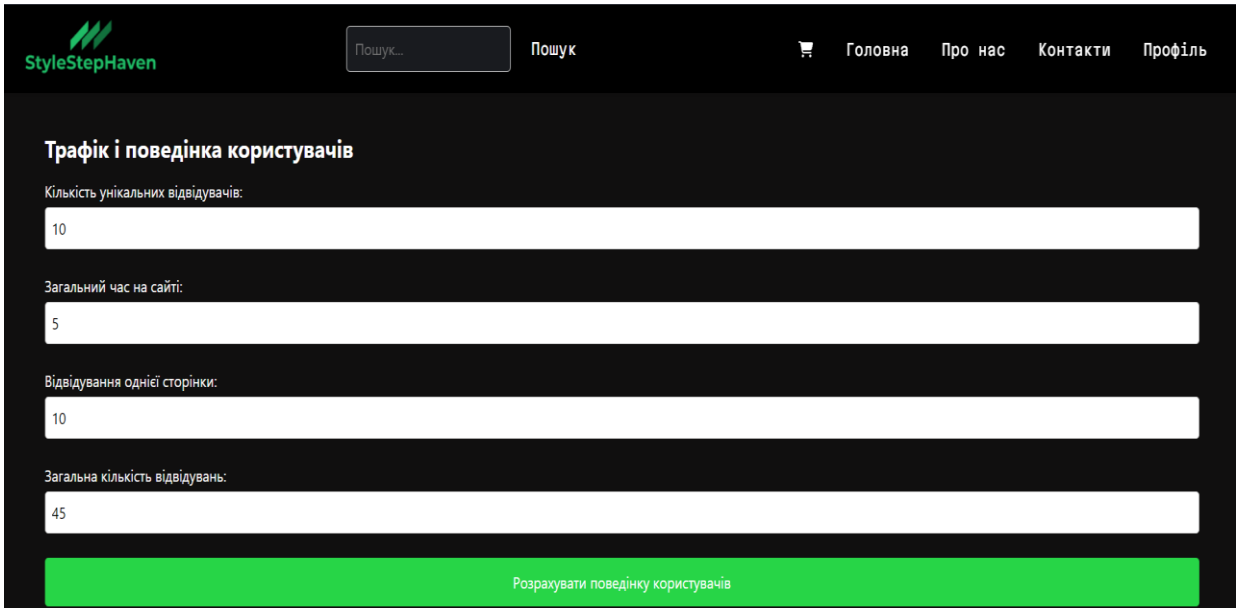

Рисунок 4.3 – Вводимо вхідні дані та натискаємо кнопку «Розрахувати поведінку користувачів»

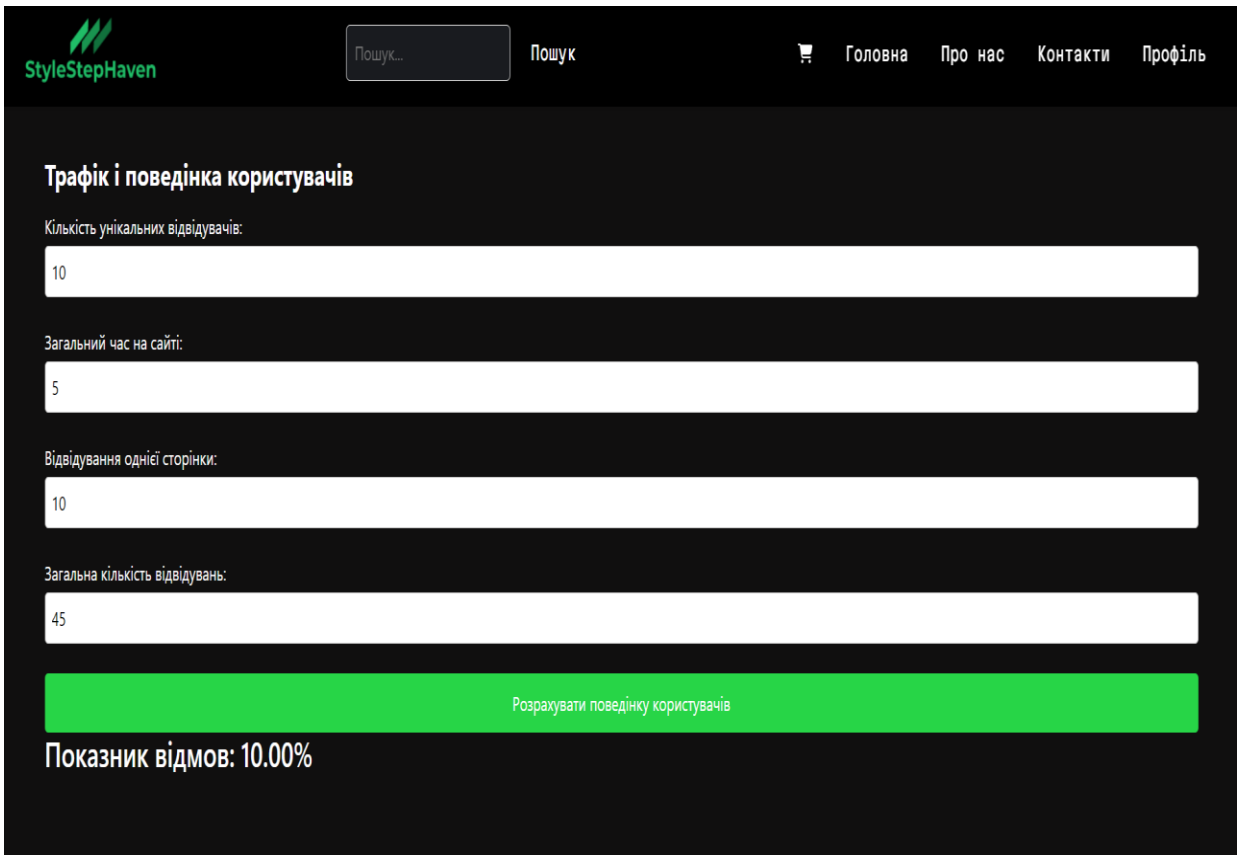

Рисунок 4.4 – Результат розрахунків

4.3 Математична модель розподілу навантаження

Модель описує розподіл навантаження на сервер, який визначається на основі кількості активних користувачів та навантаження від кожного з них. Це важливий аспект для забезпечення стабільної роботи вебсайту, особливо під час пікових навантажень.

Кількість активних користувачів (ActiveUsers):

$$\text{ActiveUsers} = \sum_{i=1}^{N} u_i \quad ,, \tag{4.3}$$

де u_i — кількість активних користувачів у момент часу i.

Навантаження від одного користувача (UserLoad):

UserLoad

Загальне навантаження на сервер (ServerLoad):

$$\text{ServerLoad} = \text{ActiveUsers} \times \text{UserLoad} \tag{4.4}$$

Як це реалізовано у коді

1. Визначення станів:

У React-компоненті визначені стани для зберігання даних про активних користувачів та навантаження:

```
const [activeUsers, setActiveUsers] = useState<number[]>([]);
const [userLoad, setUserLoad] = useState<number | "">("");
const [serverLoad, setServerLoad] = useState<number | "">("");
```

2. Обробка вводу даних:

Дані про активних користувачів вводяться у формі як кома-відокремлений список, а навантаження від одного користувача — як числове значення:

```
<label className={styles.label}>
```

Кількість активних користувачів в моменти часу (через кому):

<input type="text" name="activeUsers" className={styles.input} value={activeUsers.join(",")} onChange={(e) => setActiveUsers(e.target.value.split(",").map(Number))} />

</label>

<label className={styles.label}>

Навантаження від одного користувача:

<input type="number" name="userLoad" className={styles.input} value={userLoad} onChange={(e) => setUserLoad(parseInt(e.target.value) || "")} />

</label>

3. Функція для розрахунку кількості активних користувачів:

Функція `calculateActiveUsers` розраховує загальну кількість активних користувачів як суму масиву значень:

```
const calculateActiveUsers = (times: number[]) => {
  if (times.length === 0) return 0;
  return times.reduce((acc, user) => acc + user, 0);
};
```

4. Функція для розрахунку загального навантаження на сервер:

Функція `calculateServerLoad` розраховує загальне навантаження на сервер як добуток загальної кількості активних користувачів та навантаження від одного користувача:

```
const calculateServerLoad = (activeUsers: number, userLoad: number) => {
  return activeUsers * userLoad;
};
```

5. Обробка обчислення навантаження на сервер:

Функція `handleLoadCalculate` викликає функції розрахунку активних користувачів та загального навантаження, а потім оновлює стан `serverLoad`:

```
const handleLoadCalculate = () => {
  if (Array.isArray(activeUsers) && activeUsers.length > 0 && typeof userLoad === "number" && userLoad > 0) {
```

```
setServerLoad(calculateServerLoad(calculateActiveUsers(activeUsers), userLoad));
            } else {
                alert("Будь ласка, введіть дійсні дані для активних користувачів та навантаження від одного користувача");
            }
        };
```

6. Відображення результату:

Результат розрахунку загального навантаження на сервер відображається у компоненті, якщо значення `serverLoad` не є порожнім:

```
{serverLoad !== "" && (
    <div className={styles.result}>
        <h3>Навантаження на сервер: {serverLoad.toFixed(2)}</h3>
    </div>
)}
```

Скріншоти роботи програми (рис.4.5 та рис. 4.6)

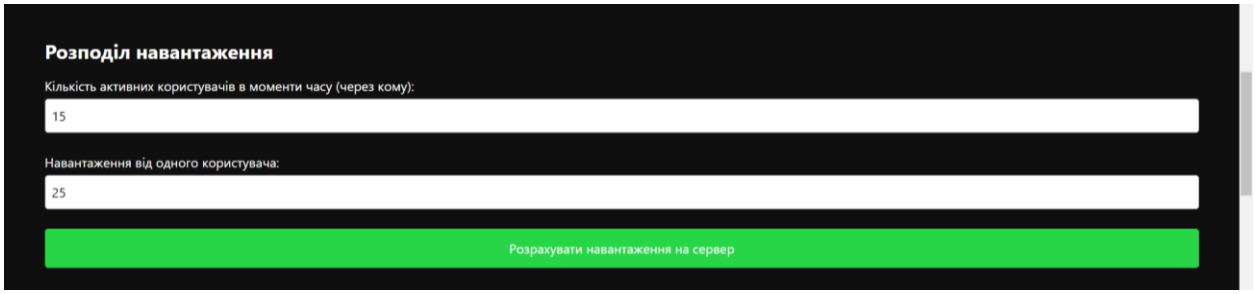

Рисунок 4.5 – Вводимо вхідні дані та натискаємо кнопку «Розрахувати навантаження на сервер»

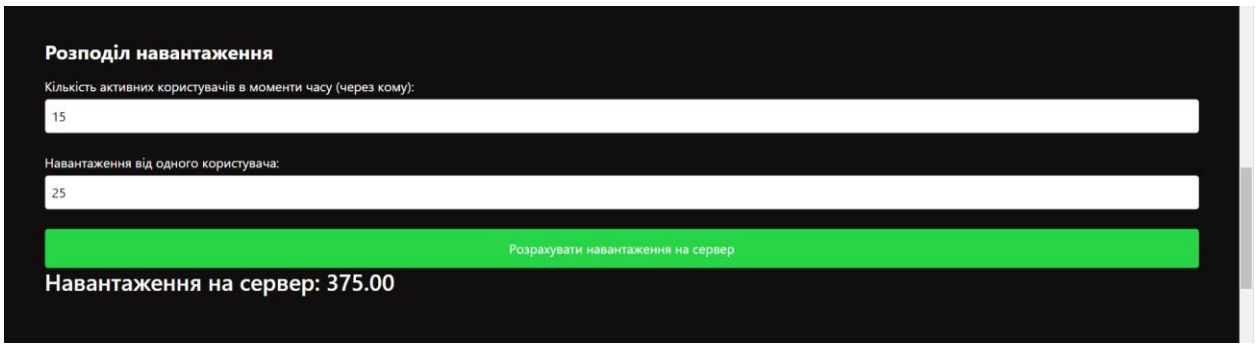

Рисунок 4.6 – Результат розрахунків

4.4 Математична модель аналізу користувацької поведінки

Марковська модель використовується для визначення ймовірностей переходу користувача між різними сторінками сайту. В цій моделі ймовірність переходу зі сторінки i на сторінку j представлена як:

$$P_{ij} = P(X_{t+1} = j \mid X_t = i) \tag{4.5}$$

де P_{ij} – ймовірність переходу зі сторінки i на сторінку j,

X_t – стан (сторінка) в момент часу t,

X_{t+1} – стан (сторінка) в момент часу t+1.

Цей код написаний на TypeScript і виконує обчислення ймовірностей переходу між станами на основі матриці переходів. Ось детальніше пояснення кожної частини коду:

Функція `calculateTransitionProbabilities`

```
const calculateTransitionProbabilities = (matrix: number[][], currentState: number) => {
    // Перевіряємо, чи матриця не порожня і чи поточний стан менший за довжину матриці
    if (matrix.length === 0 || currentState >= matrix.length) return [];
    // Повертаємо рядок матриці, що відповідає поточному стану
    return matrix[currentState];
};
```

1. Параметри:

- `matrix: number[][]`: Двовимірний масив (матриця) чисел, який представляє ймовірності переходів.

- `currentState: number`: Поточний стан (індекс у матриці), для якого потрібно знайти ймовірності переходів.

2. Перевірка: Функція спочатку перевіряє, чи матриця не порожня і чи поточний стан менший за довжину матриці. Якщо будь-яка з цих умов не виконується, функція повертає порожній масив.

3. Повернення: Якщо перевірка пройшла успішно, функція повертає рядок матриці, що відповідає поточному стану. Цей рядок містить ймовірності переходів з поточного стану в інші стани.

Функція `handleTransitionCalculate`

```
const handleTransitionCalculate = () => {
  // Перевіряємо, чи матриця переходів не порожня, чи поточний стан є числом, і чи поточний стан є валідним індексом у матриці
  if (transitionMatrix.length > 0 && typeof currentState === "number" && currentState >= 0 && currentState < transitionMatrix.length) {
    // Обчислюємо ймовірності переходу для поточного стану
    const probabilities = calculateTransitionProbabilities(transitionMatrix, currentState);
    // Оновлюємо стан для збереження ймовірностей переходу
    setNextStateProbabilities(probabilities);
  } else {
    // Виводимо повідомлення про помилку, якщо дані некоректні
    alert("Будь ласка, введіть дійсну матрицю ймовірностей переходу та поточний стан");
  }
};
```

1. Перевірка:

- Функція перевіряє, чи матриця переходів (`transitionMatrix`) не порожня.

- Перевіряє, чи поточний стан (`currentState`) є числом.

- Перевіряє, чи поточний стан є валідним індексом у матриці (тобто знаходиться в межах довжини матриці).

2. Обчислення: Якщо всі перевірки пройшли успішно, функція викликає `calculateTransitionProbabilities` з переданою матрицею та поточним станом, щоб обчислити ймовірності переходів для поточного стану.

3. Оновлення стану: Після обчислення ймовірностей, функція викликає `setNextStateProbabilities` для оновлення стану з новими ймовірностями переходів.

4. Повідомлення про помилку: Якщо будь-яка з перевірок не пройшла, функція виводить повідомлення про помилку, вказуючи користувачу, що необхідно ввести дійсну матрицю ймовірностей переходу та поточний стан.

Скріншоти роботи програми (рис.4.7 та рис.4.8)

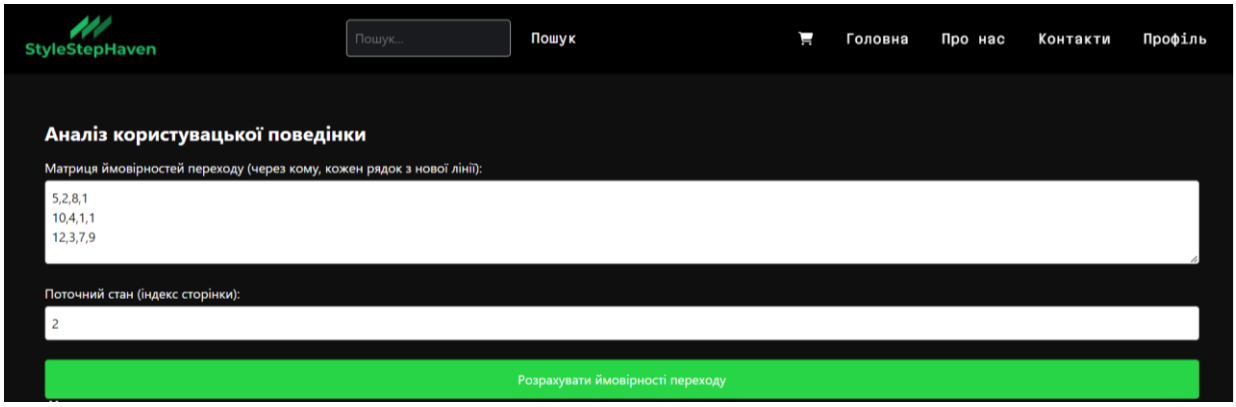

Рисунок 4.7 – Вводимо вхідні данні та натискаємо кнопку «Розрахувати ймовірності переходу

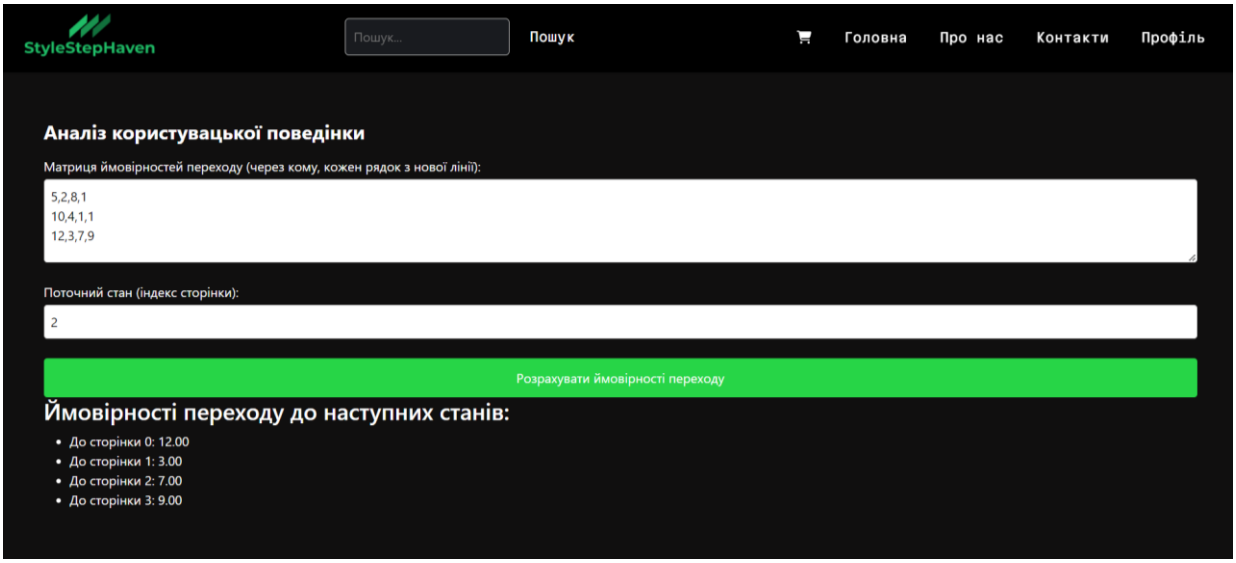

Рисунок 4.8 – Результат обчислень програми.

4.5 Математична модель швидкості завантаження

Математична модель швидкості завантаження допомагає описати, аналізувати та прогнозувати процес завантаження даних у різних мережевих і комп'ютерних системах.

4.5.1 Модель вимірювання початкового часу для завантаження комопоненту «Home»

1. Формула для обчислення часу завантаження компоненту «Home»

- `startTime = performance.now()`

2. Вимірювання кінцевого часу через невеликий проміжок часу (10 мілісекунд):

- Через 10 мс: `endTime = performance.now()`

3. Обчислення різниці між кінцевим і початковим часом:

- `loadTime = endTime - startTime`

4. Оновлення глобального стану за допомогою Redux:

- `dispatch(updateSearchDuration(loadTime))`

Отже, формула для обчислення часу завантаження сторінки виглядає так:

$$loadTime = endTime – startTime \quad (4.6)$$

де startTime - час початку вимірювання (на момент монтажу компоненту).

endTime - час завершення вимірювання (через 10 мілісекунд після монтажу компоненту).

loadTime - час завантаження компоненту, який оновлюється у глобальному стані Redux.

Ось цей процес у вигляді коду:

```
useEffect(() => {
  window.scrollTo(0, 0); // Прокручування сторінки вгору під час монтажу компонента
  const startTime = performance.now();
  setTimeout(() => {
    const endTime = performance.now();
    dispatch(updateSearchDuration(endTime - startTime));
  }, 10);
}, [dispatch]); // Порожній масив залежностей, щоб код викликався лише один раз
```

Цей код вимірює і оновлює час завантаження компоненту у глобальному стані додатку, використовуючи Redux для зберігання цього значення.

Скріншот роботи програми (рис. 4.9).

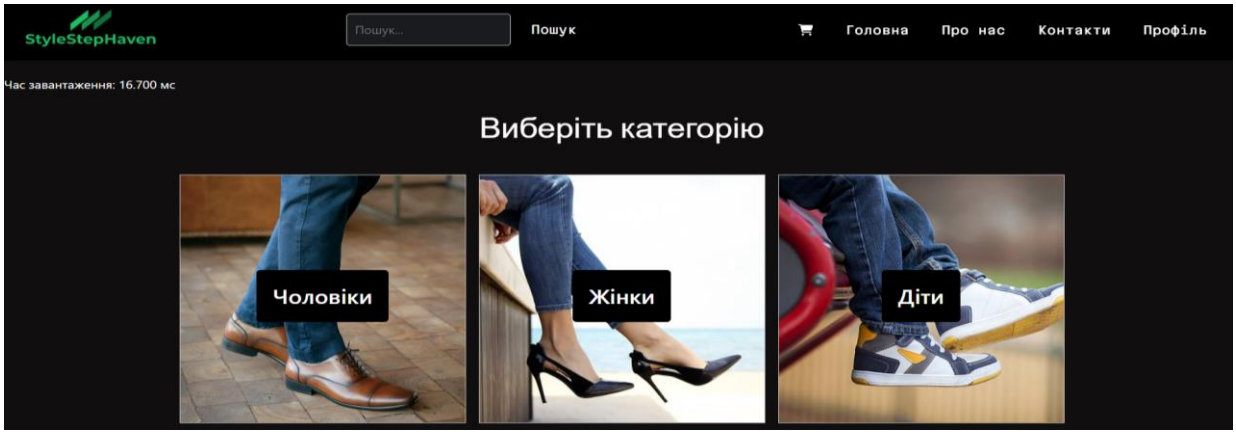

Рисунок 4.9 – Розрахунок швидкості завантаження сторінки

4.5.2 Модель «швидкості пошуку»

Формула для обчислення часу завантаження компоненту «SearchPage.

У компоненті `SearchPage` використовується Redux для збереження часу, витраченого на пошук. Цей час вимірюється під час виконання пошукового запиту та зберігається у стані Redux. Модель швидкості пошуку може бути описана наступним чином:

1. Вимірювання часу пошуку:

- Під час виконання пошукового запиту, вимірюється час початку та закінчення пошуку.

- Різниця між часом закінчення та початку пошуку визначає час пошуку.

```
const startTime = performance.now();
// виконання пошукового запиту
const endTime = performance.now();
const searchDuration = endTime - startTime;
dispatch(updateSearchDuration(searchDuration));
```

2. Збереження часу пошуку:

- Значення `searchDuration` зберігається у стані Redux за допомогою екшену `updateSearchDuration`.

3. Відображення часу пошуку:

- Компонент `SearchPage` отримує `searchDuration` з Redux і відображає його на сторінці, відформатованим з трьома знаками після коми:

```
<p>Час пошуку: {searchDuration.toFixed(3)} мс</p>
```

Це забезпечує користувачам зворотний зв'язок щодо продуктивності пошукової функції у вигляді часу, витраченого на пошук.

Скріншот роботи програми (рис. 4.10)

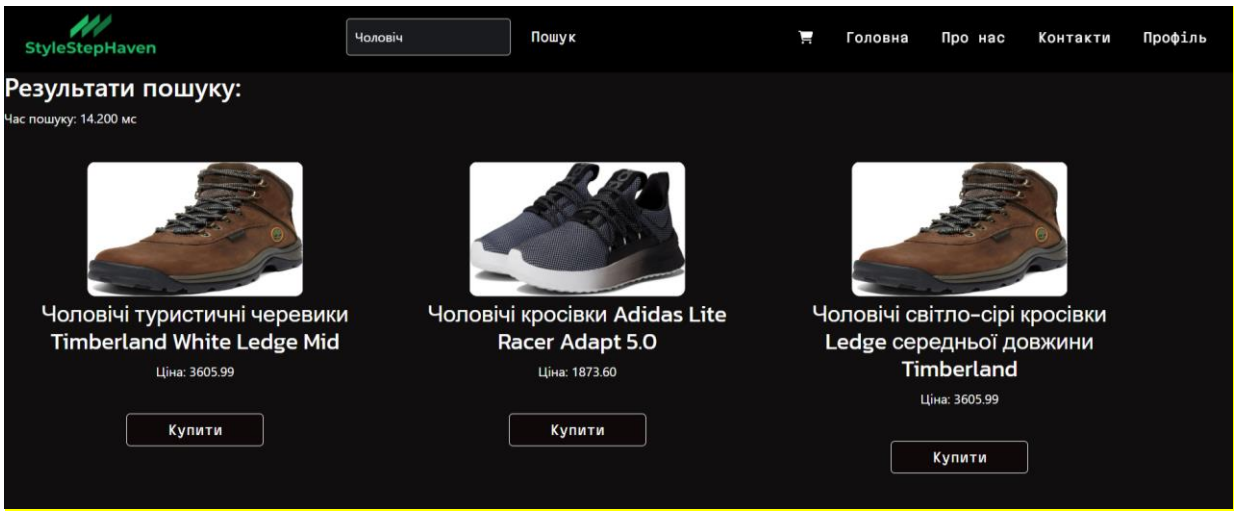

Рисунок 4.10 – Результати роботи програми

4.5.3 Модель швидкості авторизації

Формула для обчислення часу завантаження компоненту«Profile».

У компоненті `Profile` реалізована модель швидкості авторизації, яка включає наступні етапи:

1. Вимірювання часу авторизації:

- Час авторизації вимірюється під час процесу авторизації користувача, коли він входить у систему. Це може бути виконано, наприклад, під час відправки запиту на сервер для перевірки облікових даних користувача та отримання відповіді.

- Час початку та закінчення авторизації вимірюється за допомогою `performance.now()` або подібного методу.

2. Збереження часу авторизації у URL:

- Після успішної авторизації час авторизації передається як параметр у URL.

navigate(`/profile?authTime=${authTime}`);

3. Отримання та відображення часу авторизації:

- Компонент `Profile` витягує параметр `authTime` з URL під час завантаження сторінки.

- Якщо параметр `authTime` існує, він зберігається в стані `authTime` і відображається на сторінці профілю.

{authTime !== null && Час авторизації: {authTime} мс }

Ця модель дозволяє відстежувати та відображати час, витрачений на авторизацію користувача, надаючи зворотний зв'язок щодо продуктивності процесу авторизації.

Скріншот роботи програми (рис. 4.11).

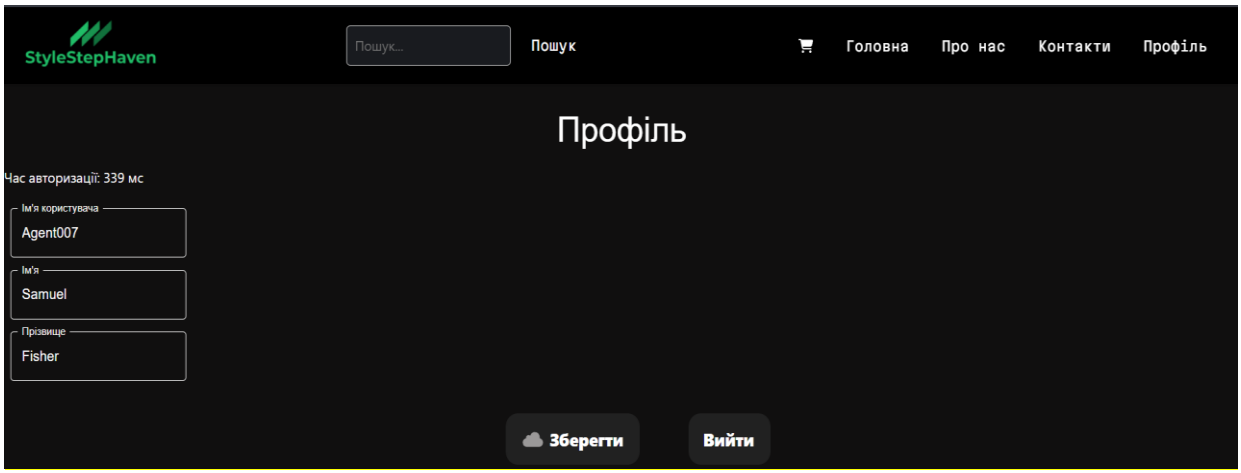

Рисунок 4.11 – Результати роботи програми

4.6 Висновки за розділом

Розділ "Математичні моделі" демонструє комплексний підхід до аналізу та оптимізації різних аспектів роботи веб-ресурсів. Використання математичних моделей дозволяє не тільки глибше зрозуміти поведінку користувачів і навантаження на сервери, але й впроваджувати ефективні рішення для покращення продуктивності та користувацького досвіду.

5 ТЕСТУВАННЯ

5.1 Тестування категорій

При переході користувача на сайт він бачить головну сторінку. На цій сторінці представлені три основні категорії: "Чоловіки", "Жінки" та "Діти" (рис. 5.1).

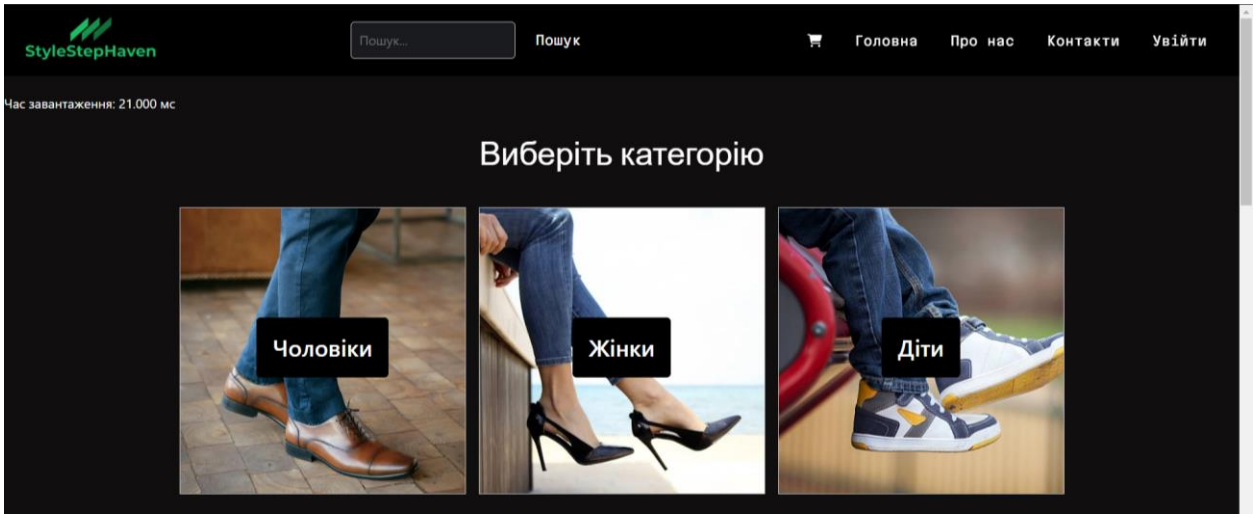

Рисунок 5.1 – Головна сторінка з категоріями

Далі розглянемо і протестуємо роботу кожної з цих категорій.

Категорія «Чоловіки».

Ця категорія призначена для відображення товарів, які орієнтовані на чоловічу аудиторію. При переході в цю категорію бачимо тільки чоловічі товари (рис. 5.2).

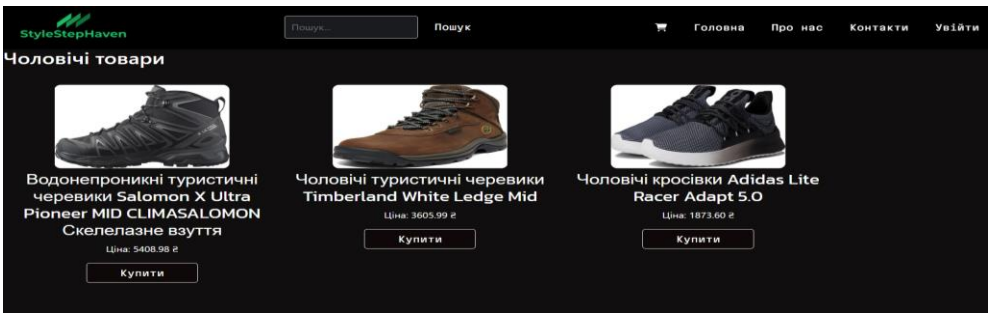

Рисунок 5.2 – Категорія товарів «Чоловіки»

Категорія «Жінки».

Ця категорія орієнтована на жіночу аудиторію і включає товари, такі як взуття. При переході в цю категорію бачимо жіночі товари (рис. 5.3).

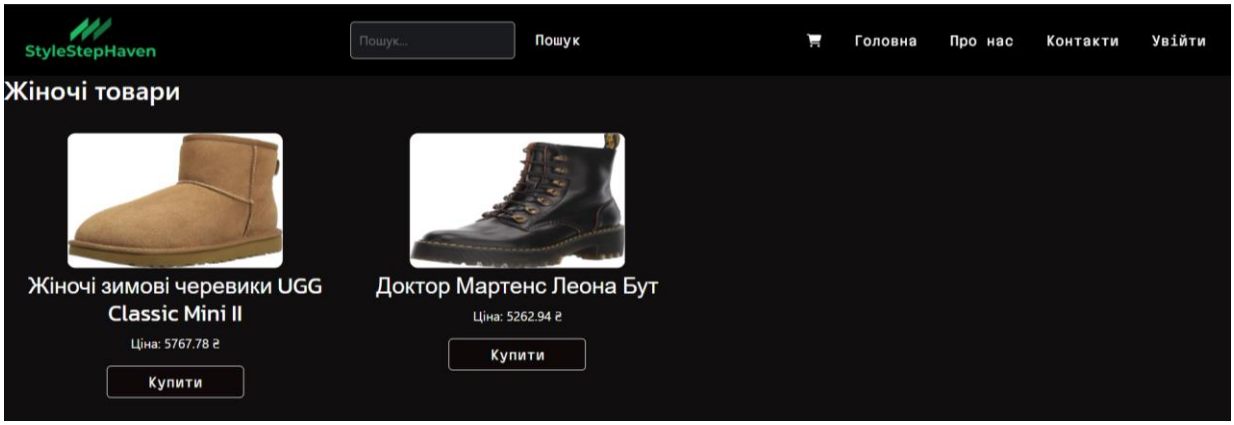

Рисунок 5.3 – Категорія товарів «Жінки»

Категорія «Діти».

Категорія «Діти» містить товари для дітей, включаючи взуття (рис. 5.4).

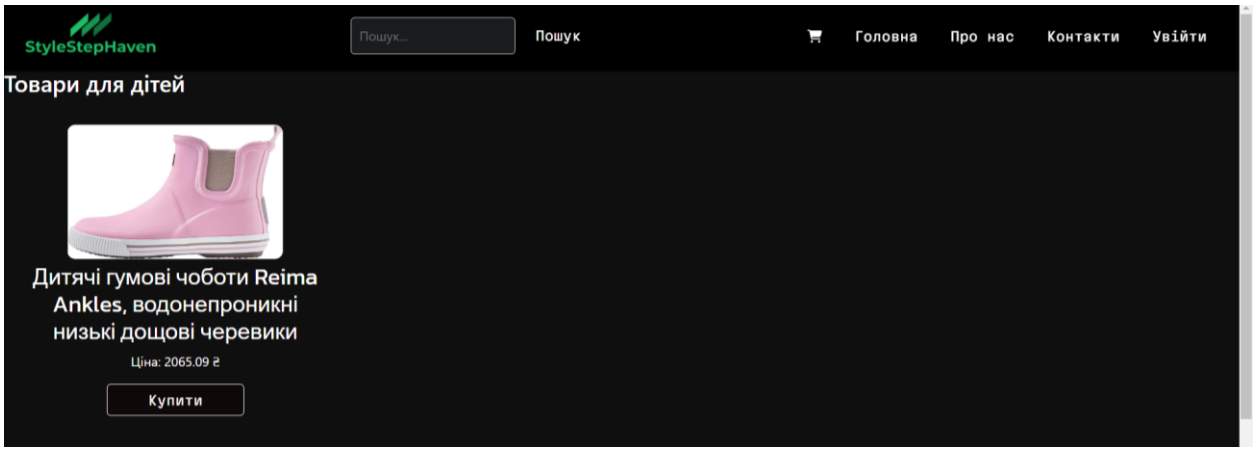

Рисунок 5.4 – Категорія товарів «Діти»

5.2 Тестування покупки товарів

На головній сторінці користувач натискає на будь-який товар і переходить до його сторінки (рис. 5.5 та рис. 5.6).

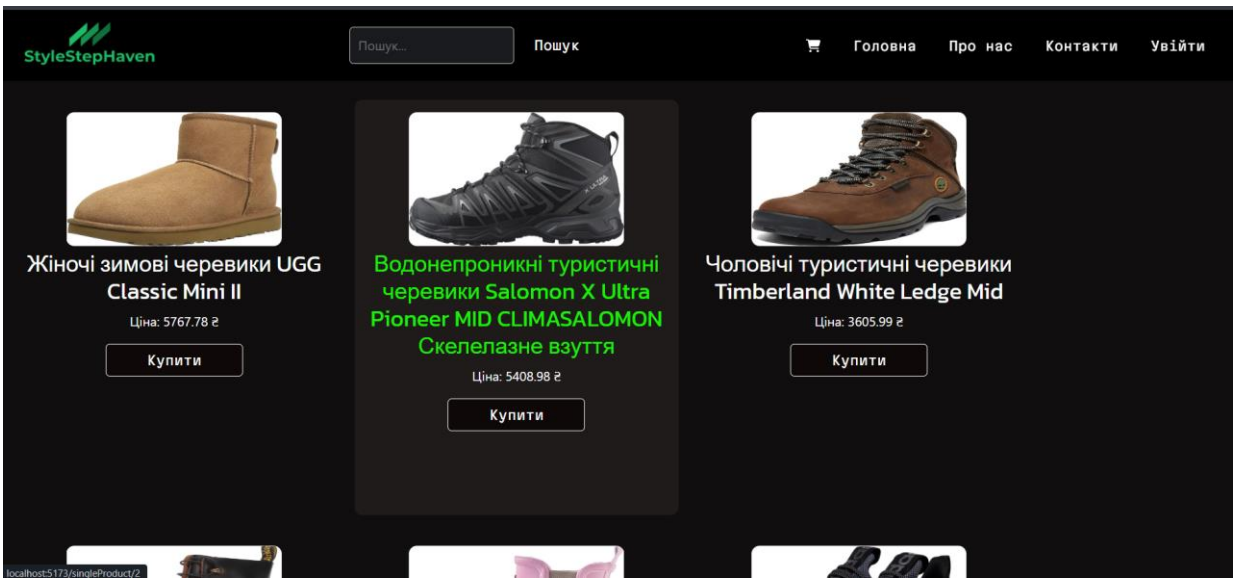

Рисунок 5.5 – Сторінка вибору товарів

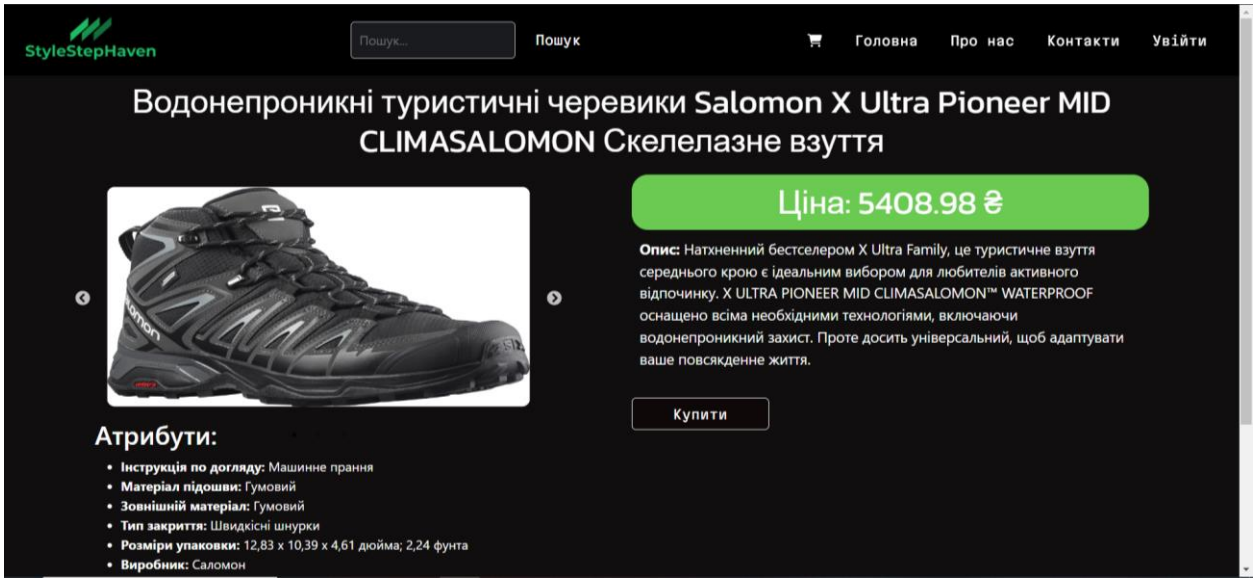

Рисунок 5.6 – Сторінка обраного товару

На цій сторінці користувач бачить назву, зображення, опис, ціну та атрибути товару. Натискаємо кнопку купити и товар додається у кошик (рис. 5.7).

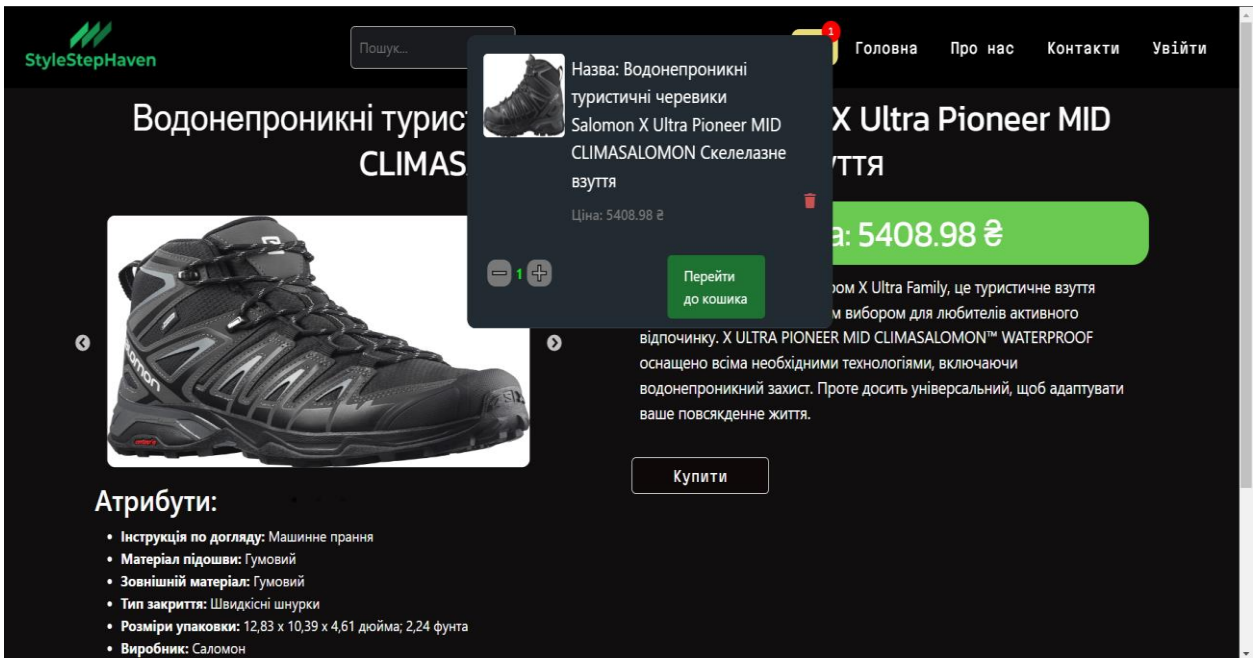

Рисунок 5.7 – Товар доданий до кошика

Після цього користувач переходить до кошика та бачить сторінку оформлення замовлення. На цій сторінці необхідно внести контактні дані, зокрема ім'я, номер телефону та місто. Далі користувач вибирає метод доставки та спосіб оплати (рис. 5.8).

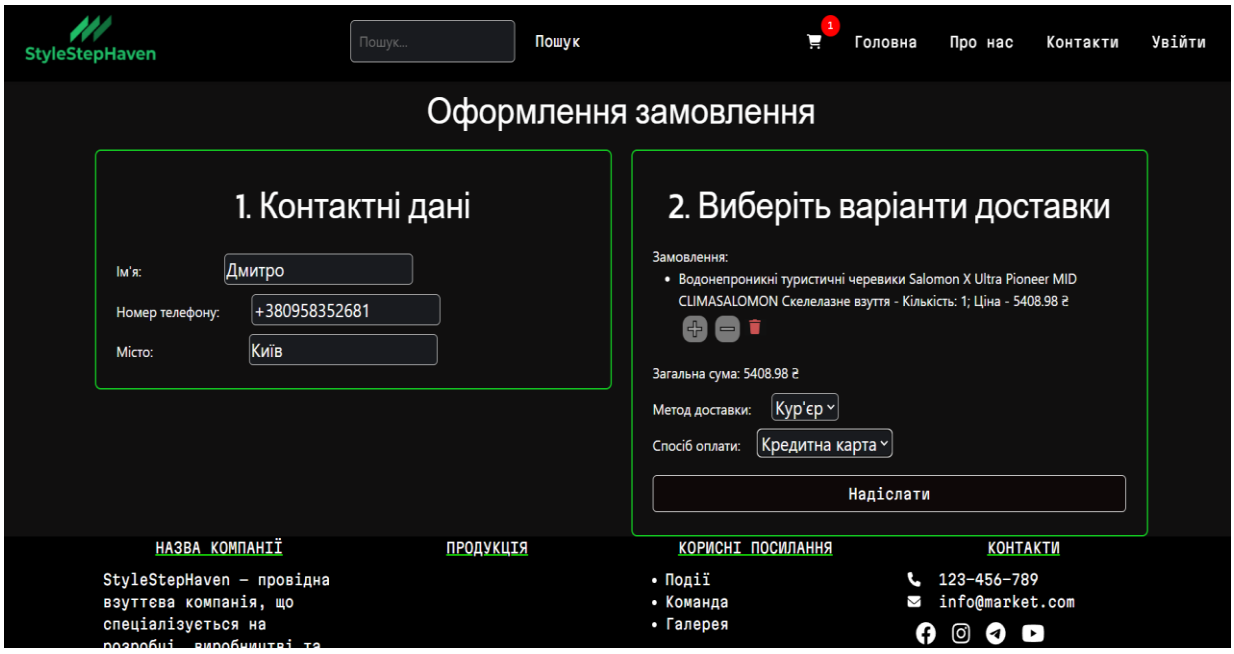

Рисунок 5.8 – Сторінка оформлення замовлення

Після цього користувач натискає кнопку "Надіслати" і переходить до сторінки оплати. На цій сторінці він вводить дані своєї картки (рис. 5.9).

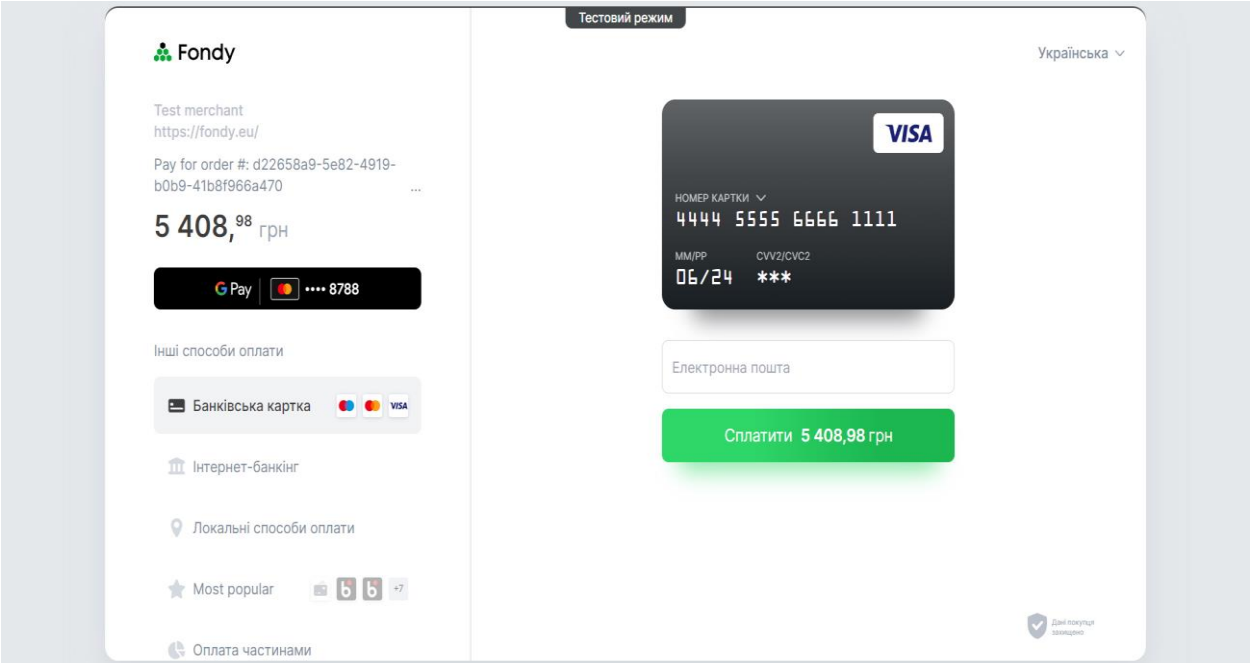

Рисунок 5.9 – Сторінка оплати товару

5.3 Тестування пошуку товарів

Пошук знаходиться у хедері. Користувач вводить у нього дані та перевіряє роботу функції пошуку (рис. 5.10).

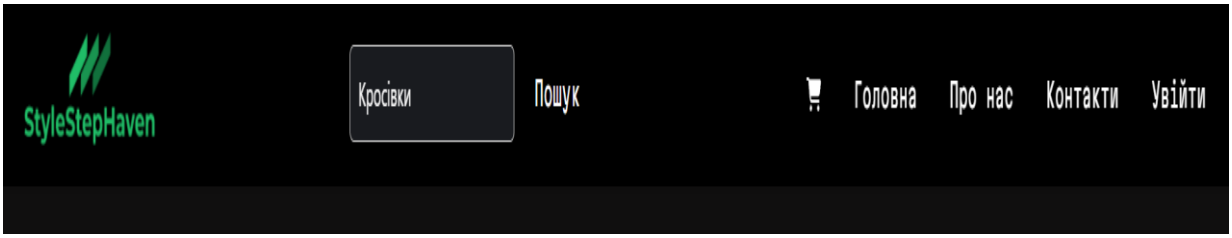

Рисунок 5.10 – Вводимо дані в поле пошуку

Користувач натискає кнопку «Пошук», після чого на екрані з'являються результати пошуку. Система обробляє введені дані та виводить відповідні товари або інформацію, що відповідає запиту користувача (рис. 5.11).

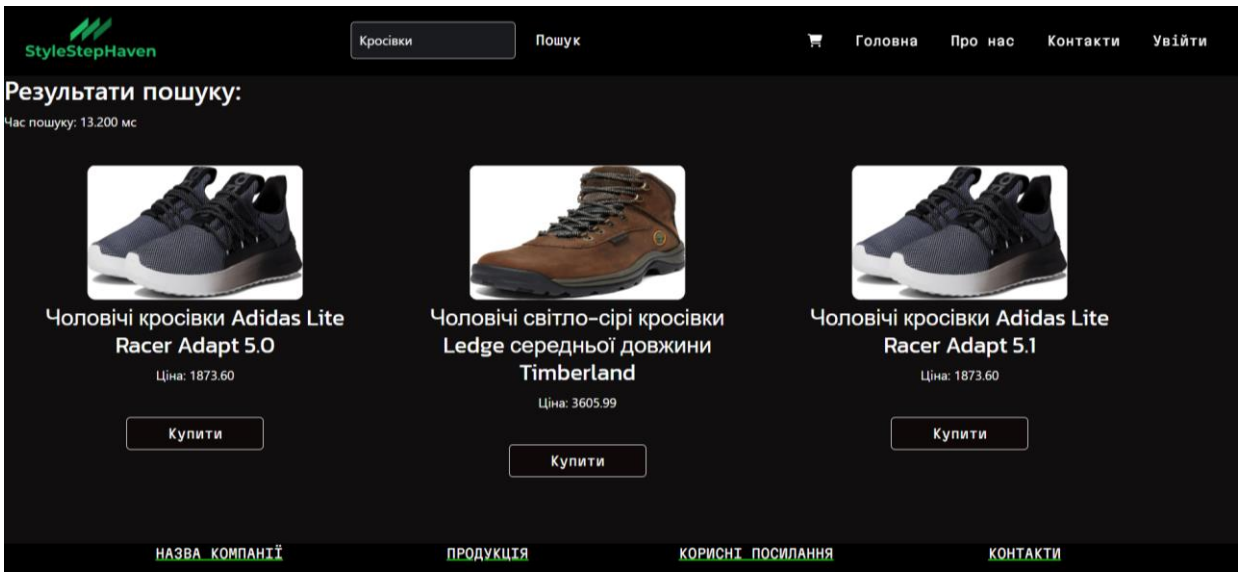

Рисунок 5.11 – Результати пошуку

5.4 Тестування реєстрації та авторизації

На сторінці реєстрації користувач заповнює поля, такі як ім'я, прізвище, логін та пароль. Кожне з цих полів є обов'язковим для успішного створення облікового запису(рис. 5.12).

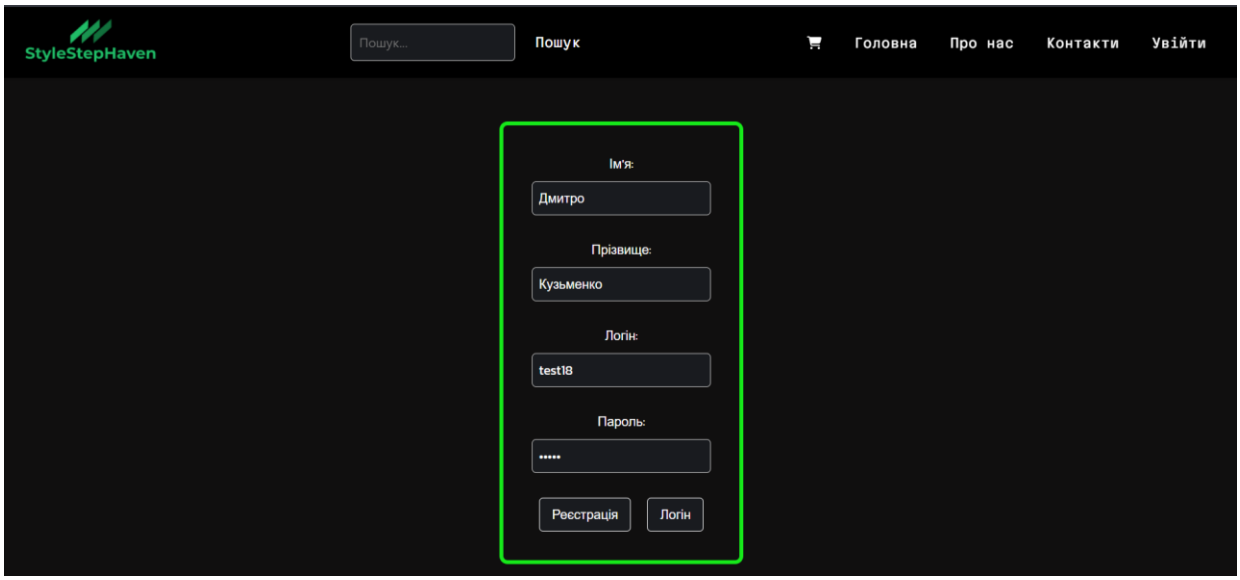

Рисунок 5.12 – Сторінка реєстрації с заповненими полями

Після введення необхідних даних користувач натискає кнопку «Реєстрація». Після цього на екрані з'являється модальне вікно з

повідомленням «Реєстрація успішна!». Це підтверджує успішне створення облікового запису (рис. 5.13).

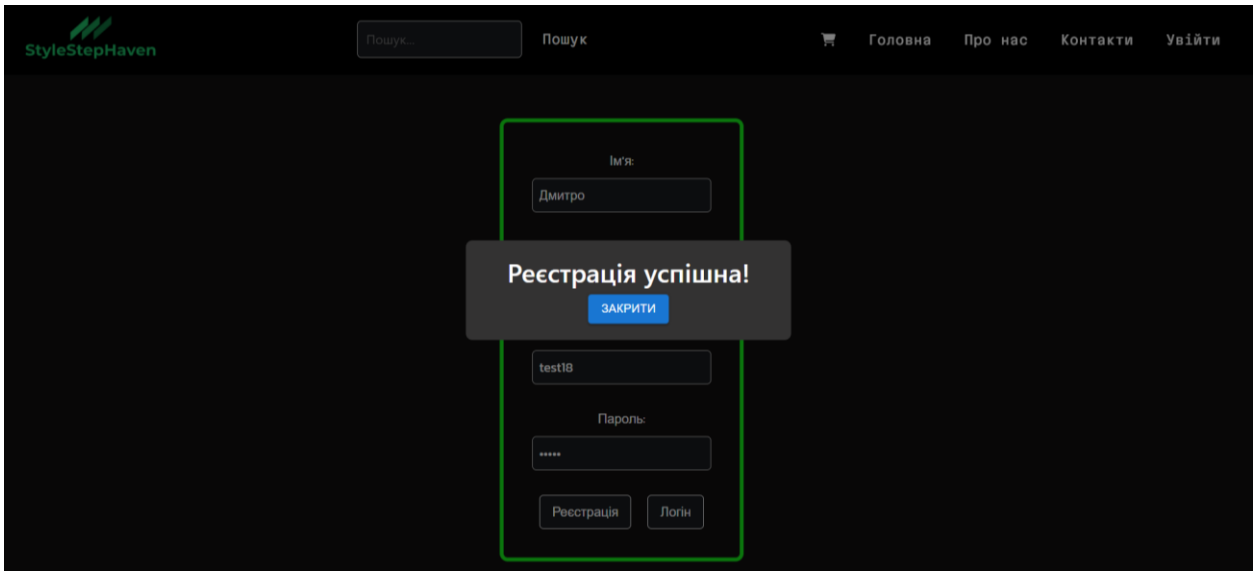

Рисунок 5.13 – Повідомлення про успішну реєстрацію

Після успішної реєстрації користувач переходить на сторінку авторизації. Там він вводить дані, які вказував при реєстрації, а саме логін та пароль (рис. 5.14).

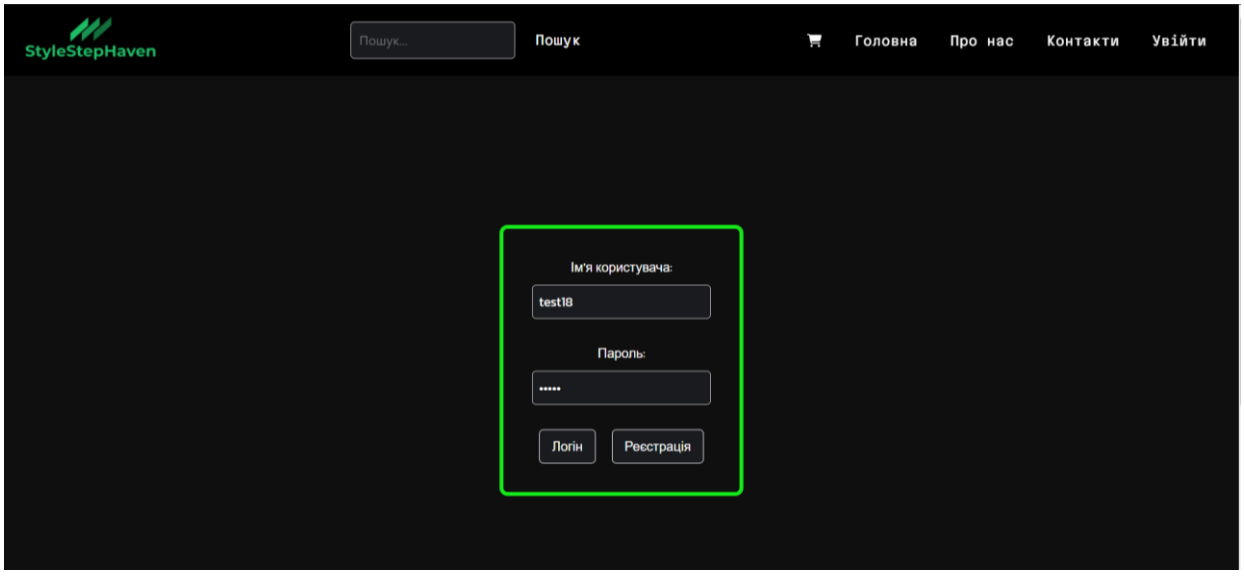

Рисунок 5.14 – Сторінка авторизації

Після введення даних користувач натискає кнопку "Логін". Після цього він переходить на сторінку профілю, де відображається інформація про користувача(рис. 5.15).

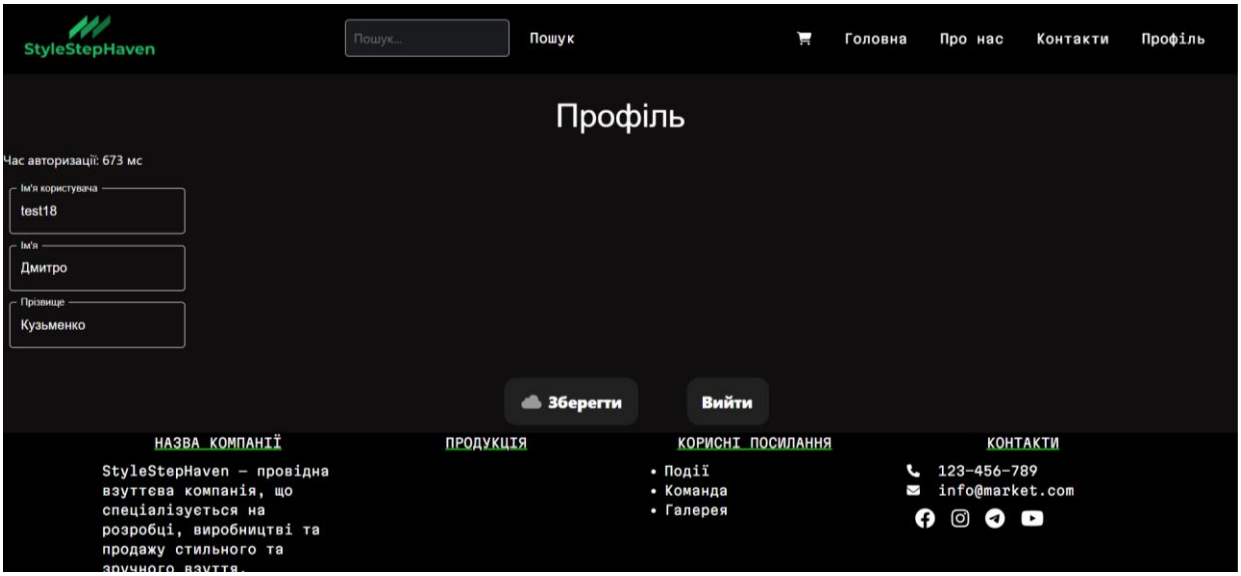

Рисунок 5.15 – Сторінка профілю користувача

5.5 Висновок за розділом

У цих розділах ми детально розглянули основні функції та процеси, пов'язані з використанням веб-сайту. Від перегляду товарів до оформлення замовлення, від реєстрації користувачів до входу в обліковий запис, кожен етап взаємодії з сайтом був розглянутий з урахуванням користувальницького досвіду та функціональності. Ось деякі ключові аспекти, що були протестовані:

1. Навігація та пошук: Важливість швидкого та зручного пошуку товарів на сайті. Розміщення кнопок та елементів навігації грає важливу роль у полегшенні користувачам доступу до потрібної інформації.

2. Реєстрація та авторизація: Процеси створення облікового запису та входу в систему повинні бути інтуїтивно зрозумілими та ефективними для користувачів.

3. Кошик та оформлення замовлення: Важливість зручного та простого процесу додавання товарів до кошика та оформлення замовлення не може бути

недооціненою. Користувачі повинні мати можливість легко контролювати свої покупки та швидко робити оплату.

4. Профіль користувача: Важливо, щоб сторінка профілю надавала користувачеві зручний доступ до основної інформації про нього та його активності на сайті.

5. Повідомлення про стан операцій: Важливо мати ефективну систему сповіщень та підтверджень, щоб користувачі завжди були в курсі стану своїх операцій, таких як реєстрація, вхід, оформлення замовлення та інше.

В цілому, створення зручного та інтуїтивно зрозумілого користувальницького досвіду є ключовим аспектом розробки веб-сайтів. Послідовне вдосконалення цих процесів допомагає підвищити задоволення користувачів від використання сайту та сприяє збільшенню конверсії та лояльності клієнтів.

ВИСНОВКИ

Під час роботи розбирається обрана тема, визначається об’єкт і тема дослідження, з’ясовується мета роботи. Вибираються засоби, інструменти та прийоми, які використовуються для розробки програми, а саме:

- Фреймворк розробки на стороні клієнта: React.js;
- Фреймворк розробки серверної частини: Django;
- Фреймворк реалізації REST API: Django Rest API;
- ORM система взаємодії з базами даних: Django ORM.

Проведено аналіз стану інтернет-магазинів для покупки взуття, виконано порівняльний аналіз аналогів та розглянуто методи розв’язання задачі. Визначено постановку задач, що стоять перед розробниками, і зроблено відповідні висновки.

Розроблено метод і алгоритми роботи системи, а також веб-інтерфейс користувача, що забезпечує зручність та ефективність взаємодії з користувачами. Зроблено висновки щодо ефективності запропонованих рішень.

Проведено варіантний аналіз та обґрунтовано вибір засобів для реалізації програмного засобу. Обрано MVC-архітектурний шаблон додатку та фреймворки Django і React для розробки серверної та клієнтської частин відповідно. Зроблено висновки про переваги використаних технологій.

Описано основну структуру клієнтської частини додатку, включаючи огляд компонентів "Basket", "About", "Contacts", "Registration" та "Login". Зроблено висновки щодо структури та функціональності цих компонентів.

Розглянуто різні математичні моделі, що використовуються для аналізу та оптимізації роботи веб-ресурсів:

- модель кількості відвідувачів у різні сезони року;
- модель трафіку і поведінки користувачів;
- модель розподілу навантаження;

- модель аналізу користувацької поведінки;

- модель швидкості завантаження (зокрема для компонентів "Home", "SearchPage" та "Profile").

Зроблено висновки щодо ефективності застосування цих моделей для підвищення продуктивності та покращення користувацького досвіду.

Проведено тестування різних аспектів роботи інтернет-магазину:

- тестування категорій;

- тестування покупки товарів;

- тестування пошуку товарів;

- тестування реєстрації та авторизації.

Основним напрямком є розвиток інтернет-магазинів з використанням веб-технологій, зокрема створення веб-серверів для забезпечення взаємодії з користувачами та впровадження REST API для обміну даними з клієнтськими додатками. Використання JavaScript, Python і редактора коду VSCode сприяє ефективній розробці та тестуванню програмного забезпечення, що забезпечує високу якість кінцевого продукту.

Наукова новизна та практична значимість отриманих результатів полягають у впровадженні передових методів розробки інтернет-магазинів, що базуються на використанні сучасних веб-технологій та математичних моделей. Підхід, що використовує фреймворки React.js та Django, дозволяє створювати динамічні та ефективні інтерфейси для користувачів, а також надійні серверні застосунки з підтримкою REST API для взаємодії з клієнтськими додатками.

Застосування математичних моделей, таких як моделі кількості відвідувачів у різні сезони року, моделі трафіку і поведінки користувачів, моделі розподілу навантаження та інші, дозволяє аналізувати та оптимізувати роботу інтернет-магазинів, сприяючи підвищенню їх продуктивності та покращенню користувацького досвіду.

Результати дослідження та розробки інтернет-магазинів мають велику практичну цінність для бізнесу, оскільки вони дозволяють підвищити

конкурентоспроможність підприємства, забезпечити задоволеність користувачів та оптимізувати витрати на його підтримку та розвиток.

ПЕРЕЛІК ПОСИЛАНЬ

1. Магазин Розетка. URL: https://rozetka.com.ua/ua/.
2. Магазин Sezon. URL: https://sezon.ua/.
3. Магазин Foxtrot. URL: https://www.foxtrot.com.ua/.
4. Матеріал з Вікіпедії про HTML. URL: https://uk.wikipedia.org/wiki/HTML.
5. Матеріал з Вікіпедії про CSS. URL: https://uk.wikipedia.org/wiki/CSS.
6. Матеріал з Вікіпедії про JavaScript. URL: https://uk.wikipedia.org/wiki/JavaScript.
7. Матеріал про TypeScript . URL: https://www.typescriptlang.org/.
8. Матеріал з Вікіпедії про Redux. URL: https://uk.wikipedia.org/wiki/Redux.
9. Матеріал про Python. URL: https://www.python.org/.
10. Матеріал про React. URL: https://react.dev/.
11. Матеріал про Django. URL: https://www.djangoproject.com/).
12. Матеріал про MVC . URL: http://surl.li/udxtp.
13. Матеріал про Django REST framework . URL: https://www.django-rest-framework.org/.
14. Роббінс Дж. Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics. O'Reilly Media, 2018. 808с. URL: https://a.co/d/0FUveIa.
15. Node.js Design Patterns. Packt Publishing, 2014. 454с. URL: https://a.co/d/5P4YQEj.
16. Круг С. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. New Riders, 2014. 216с. URL: https://a.co/d/6BiBdYP
17. Османі Е. Learning JavaScript Design Patterns. O'Reilly Media, 2012. 254 с. URL: https://a.co/d/abgw0es.

18. Мерфі P. jQuery Fundamentals. Bocoup, LLC, 2011. 150с. URL:https://books.google.com.ua/books/about/jQuery_Fundamentals.html?id=DYV7AgAAQBAJ&redir_esc=y.

19. Фаулер M. Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2002. 557 с. URL: https://a.co/d/aBlLldu.

20. Ньюман C. Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2015. 280 с. URL: https://a.co/d/6sNoLDo.

ДОДАТОК А
Зауваження нормоконтролера

Таблиця А.1

Сторінка	Позначення	Зміст зауваження
С.8-9, 12-18, 38-40,41-47, 51-56, 67-68		Невірно оформлені переліки
Розділ 4		Формули та змінні зроблено як рисунки, а не в редакторі формул
С. 42-45, розділ 4		Програмний код повинен бути оформлений як рисунок
ПЗ		Нещільне заповнення сторінок ПЗ

ДОДАТОК Б
Текст програмного застосунку

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN"
crossorigin="anonymous" />
    <link
href="https://fonts.googleapis.com/css2?family=Comfortaa:wght@300;400;500;600;700&family=Martia
n+Mono&family=Poppins:wght@300;400;500;600;700;800&family=Source+Sans+3&display=swap"
rel="stylesheet" />
    <link rel="preconnect" href="https://fonts.googleapis.com" />
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin />
    <link
href="https://fonts.googleapis.com/css2?family=Comfortaa:wght@300;400;500;600;700&family=Kanit
&family=Martian+Mono&family=Poppins:wght@300;400;500;600;700;800&family=Source+Sans+3&d
isplay=swap" rel="stylesheet" />

    <title>StyleStepHaven - shoe store</title>

    <title>StyleStepHaven</title>
    <meta name="description" content="In this store you can find shoes for the whole family"
/>
  </head>
  <body>
    <div id="root"></div>
    <script src="https://pay.fondy.eu/static_common/v1/checkout/ipsp.js"></script>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

Main.tsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
import { Provider } from "react-redux";
import { store } from "./store/store";
import "./global.css";

const rootElement = document.getElementById("root");

if (rootElement) {
   const root = ReactDOM.createRoot(rootElement);

   root.render(
      <React.StrictMode>
         <Provider store={store}>
            <App />
         </Provider>
      </React.StrictMode>
   );
} else {
   console.error("Root element not found");
}
```

Global.css

```
body {
   background-color: #100f0f;
   color: #fff;
}
h1 {
   text-align: center;
   padding: 10px;
   font-family: Kanit, sans-serif;
}
```

App.tsx

```
// App.tsx
```

```
import { useState, useEffect } from "react";
import { BrowserRouter, Route, Routes } from "react-router-dom";
import Home from "./components/home/Home";
import About from "./components/about/About";
import Contacts from "./components/contacts/Contacts";
import NotFound from "./components/NotFound";
import Header from "./components/header/Header";
import HeaderAuth from "./components/header/HeaderAuth"; // Импортируем HeaderAuth
import Profile from "./components/profile/Profile";
import Login from "./components/login/Login";
import Basket from "./components/basket/Basket";
import Footer from "./components/footer/Footer";
import Registration from "./components/login/Registration";
import SingleProduct from "./components/product/SingleProduct";
import Men from "./components/catalog/Men";
import Women from "./components/catalog/Women";
import Kids from "./components/catalog/Kids";
import SearchPage from "./components/searchPage/SearchPage";
import styles from "./App.module.css";

function App() {
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  const [hasAccessToken, setHasAccessToken] = useState(false);

  useEffect(() => {
    // Перевіряємо наявність access token в localStorage під час завантаження програми
    const accessToken = localStorage.getItem("access_token");
    if (accessToken) {
      setIsLoggedIn(true);
      setHasAccessToken(true);
    }
  }, []);

  return (
    <div className={styles.App}>
      <BrowserRouter>
        {/* Відображаємо HeaderAuth, якщо немає access token, та Header, якщо є access
token */}
```

```
                {hasAccessToken ? <HeaderAuth isLoggedIn={isLoggedIn} /> : <Header
isLoggedIn={isLoggedIn} />}
                <Routes>
                  <Route path="basket" element={<Basket />} />
                  <Route path="/" element={<Home />} />
                  <Route path="about" element={<About />} />
                  <Route path="contacts" element={<Contacts />} />
                  <Route path="login" element={<Login setIsLoggedIn={setIsLoggedIn} />} />
                  <Route path="profile" element={<Profile isLoggedIn={isLoggedIn} />} />
                  <Route path="registration" element={<Registration />} />
                  <Route path="singleProduct/:id" element={<SingleProduct />} />
                  <Route path="searchPage" element={<SearchPage />} />
                  <Route path="men" element={<Men />} />
                  <Route path="women" element={<Women />} />
                  <Route path="kids" element={<Kids />} />
                  <Route path="*" element={<NotFound />} />
                </Routes>
                <Footer />
              </BrowserRouter>
            </div>
          );
        }

        export default App;
```

about.tsx

```
// Імпорт необхідних бібліотек і стилів
import React, { useState, useEffect } from "react";
import { motion } from "framer-motion";
import styles from "./About.module.css";

// Типи властивостей компоненту (в даному випадку вони пусті)
type AboutProps = {};

// Основний компонент About
const About: React.FC<AboutProps> = () => {
  // Визначення станів для зберігання даних про відвідувачів у різні сезони
```

```
    const [summerVisitors, setSummerVisitors] = useState<number | "">(""); // Стан для зберігання кількості літніх відвідувачів
    const [winterVisitors, setWinterVisitors] = useState<number | "">(""); // Стан для зберігання кількості зимових відвідувачів
    const [springVisitors, setSpringVisitors] = useState<number | "">(""); // Стан для зберігання кількості весняних відвідувачів
    const [fallVisitors, setFallVisitors] = useState<number | "">(""); // Стан для зберігання кількості осінніх відвідувачів

    // Визначення станів для інших метрик відвідуваності
    const [uniqueVisitors, setUniqueVisitors] = useState<number | "">(""); // Стан для зберігання кількості унікальних відвідувачів
    const [averageTimeOnSite, setAverageTimeOnSite] = useState<number | "">(""); // Стан для зберігання середнього часу на сайті
    const [singlePageVisits, setSinglePageVisits] = useState<number | "">(""); // Стан для зберігання кількості відвідувань однієї сторінки
    const [totalVisits, setTotalVisits] = useState<number | "">(""); // Стан для зберігання загальної кількості відвідувань
    const [bounceRate, setBounceRate] = useState<number | "">(""); // Стан для зберігання показника відмов

    // Визначення станів для активних користувачів і навантаження на сервер
    const [activeUsers, setActiveUsers] = useState<number[]>([]); // Стан для зберігання кількості активних користувачів
    const [userLoad, setUserLoad] = useState<number | "">(""); // Стан для зберігання навантаження від одного користувача
    const [serverLoad, setServerLoad] = useState<number | "">(""); // Стан для зберігання загального навантаження на сервер

    // Визначення станів для матриці ймовірностей переходів та поточного стану
    const [transitionMatrix, setTransitionMatrix] = useState<number[][]>([]); // Стан для зберігання матриці ймовірностей переходів
    const [currentState, setCurrentState] = useState<number | "">(""); // Стан для зберігання поточного стану
    const [nextStateProbabilities, setNextStateProbabilities] = useState<number[]>([]); // Стан для зберігання ймовірностей переходу до наступних станів

    // Функція для обчислення відвідувачів на основі кількості літніх відвідувачів
```

```
    const calculateFromSummer = (summer: number) => {
      setWinterVisitors(summer * 0.6); // Обчислення зимових відвідувачів на основі літніх
      setSpringVisitors(summer * 0.8); // Обчислення весняних відвідувачів на основі літніх
      setFallVisitors(summer * 0.8); // Обчислення осінніх відвідувачів на основі літніх
    };

    // Функція для обчислення відвідувачів на основі кількості зимових відвідувачів
    const calculateFromWinter = (winter: number) => {
      const summer = winter * (10 / 6); // Обчислення літніх відвідувачів на основі зимових
      calculateFromSummer(summer); // Виклик функції обчислення для літніх відвідувачів
      setSummerVisitors(summer); // Збереження обчисленої кількості літніх відвідувачів
    };

    // Функція для обчислення відвідувачів на основі кількості весняних відвідувачів
    const calculateFromSpring = (spring: number) => {
      const summer = spring * 1.25; // Обчислення літніх відвідувачів на основі весняних
      calculateFromSummer(summer); // Виклик функції обчислення для літніх відвідувачів
      setSummerVisitors(summer); // Збереження обчисленої кількості літніх відвідувачів
    };

    // Функція для обчислення відвідувачів на основі кількості осінніх відвідувачів
    const calculateFromFall = (fall: number) => {
      const summer = fall * 1.25; // Обчислення літніх відвідувачів на основі осінніх
      calculateFromSummer(summer); // Виклик функції обчислення для літніх відвідувачів
      setSummerVisitors(summer); // Збереження обчисленої кількості літніх відвідувачів
    };

    // Функція для обробки обчислення відвідувачів
    const handleCalculate = () => {
      if (typeof summerVisitors === "number" && summerVisitors > 0) {
        calculateFromSummer(summerVisitors); // Виклик функції обчислення для літніх
відвідувачів
      } else if (typeof winterVisitors === "number" && winterVisitors > 0) {
        calculateFromWinter(winterVisitors); // Виклик функції обчислення для зимових
відвідувачів
      } else if (typeof springVisitors === "number" && springVisitors > 0) {
        calculateFromSpring(springVisitors); // Виклик функції обчислення для весняних
відвідувачів
```

```
        } else if (typeof fallVisitors === "number" && fallVisitors > 0) {
          calculateFromFall(fallVisitors); // Виклик функції обчислення для осінніх відвідувачів
        } else {
          alert("Будь ласка, введіть дійсне число відвідувачів для одного із сезонів"); // Повідомлення про помилку, якщо жодне значення не введено
        }
      };

      // Функція для обчислення середнього часу на сайті
      const calculateAverageTime = (times: number[]) => {
        if (times.length === 0) return 0; // Перевірка на порожній масив
        const total = times.reduce((acc, time) => acc + time, 0); // Підрахунок загального часу
        return total / times.length; // Розрахунок середнього часу
      };

      // Функція для обчислення показника відмов (bounce rate)
      const calculateBounceRate = (singlePageVisits: number, totalVisits: number) => {
        if (totalVisits === 0) return 0; // Перевірка на нульове значення
        return (singlePageVisits / totalVisits) * 100; // Розрахунок показника відмов
      };

      // Функція для обробки обчислення поведінки користувачів
      const handleTrafficCalculate = () => {
        if (typeof singlePageVisits === "number" && typeof totalVisits === "number") {
          setBounceRate(calculateBounceRate(singlePageVisits, totalVisits)); // Виклик функції розрахунку показника відмов та збереження результату
        } else {
          alert("Будь ласка, введіть дійсні числа для відвідувань однієї сторінки та загальних відвідувань"); // Повідомлення про помилку, якщо значення некоректні
        }
      };

      // Функція для обчислення кількості активних користувачів
      const calculateActiveUsers = (times: number[]) => {
        if (times.length === 0) return 0; // Перевірка на порожній масив
        return times.reduce((acc, user) => acc + user, 0); // Підрахунок загальної кількості активних користувачів
      };
```

```
// Функція для обчислення навантаження на сервер
const calculateServerLoad = (activeUsers: number, userLoad: number) => {
  return activeUsers * userLoad; // Розрахунок загального навантаження на сервер
};

// Функція для обробки обчислення навантаження на сервер
const handleLoadCalculate = () => {
  if (Array.isArray(activeUsers) && activeUsers.length > 0 && typeof userLoad === "number" && userLoad > 0) {
    setServerLoad(calculateServerLoad(calculateActiveUsers(activeUsers), userLoad)); // Виклик функції розрахунку навантаження на сервер та збереження результату
  } else {
    alert("Будь ласка, введіть дійсні дані для активних користувачів та навантаження від одного користувача"); // Повідомлення про помилку, якщо значення некоректні
  }
};

// Функція для обчислення ймовірностей переходу на основі матриці
const calculateTransitionProbabilities = (matrix: number[][], currentState: number) => {
  // Перевіряємо, чи матриця не порожня і чи поточний стан менший за довжину матриці
  if (matrix.length === 0 || currentState >= matrix.length) return [];
  // Повертаємо рядок матриці, що відповідає поточному стану
  return matrix[currentState];
};

// Функція для обробки обчислення ймовірностей переходу
const handleTransitionCalculate = () => {
  // Перевіряємо, чи матриця переходів не порожня, чи поточний стан є числом, і чи поточний стан є валідним індексом у матриці
  if (transitionMatrix.length > 0 && typeof currentState === "number" && currentState >= 0 && currentState < transitionMatrix.length) {
    // Обчислюємо ймовірності переходу для поточного стану
    const probabilities = calculateTransitionProbabilities(transitionMatrix, currentState);
    // Оновлюємо стан для збереження ймовірностей переходу
    setNextStateProbabilities(probabilities);
  } else {
```

```
            // Виводимо повідомлення про помилку, якщо дані некоректні
            alert("Будь ласка, введіть дійсну матрицю ймовірностей переходу та поточний стан");
        }
    };

    // Виконання дії після першого рендерингу компонента (прокрутка до початку сторінки)
    useEffect(() => {
        window.scrollTo(0, 0);
    }, []);

    // Повернення JSX для рендерингу компонента
    return (
        <div className={styles.about}>
            <motion.h1 className={styles.welcome} initial={{ x: -1000, opacity: 0 }} animate={{ x: 0, opacity: 1 }} transition={{ delay: 0.1 }}>
                Ласкаво просимо до StyleStepHaven
            </motion.h1>

            <div className={styles.about_us}>
                <motion.h2 className={styles.history}>
                    Наша історія <img src="history-book.png" className={styles.historyBookIcon} alt="History" />
                </motion.h2>
                <p className={styles.text}>Вітаємо вас у StyleStepHaven, вашому онлайн-призначенні для ексклюзивного та стильного взуття. Ми створили наш магазин, щоб надати вам найкращий вибір взуття з усього світу. Наше захоплення модою та якістю веде нас до створення унікального досвіду для наших клієнтів.</p>
                <motion.div className={styles.imageContainer} initial={{ scale: 0 }} animate={{ scale: 1 }} transition={{ delay: 0.2 }}></motion.div>

                <div className={styles.team_container}>
                    <div>
                        <h2>
                            <strong className={styles.team}>
                                Наша команда <img src="team.png" className={styles.teamIcon} alt="Team" />
                            </strong>
```

```
                </h2>
                <p className={styles.text}>StyleStepHaven - це не просто команда, це сім'я. Ми пропонуємо не тільки високоякісні продукти, але й персоналізований досвід обслуговування. Наші фахівці з взуття зроблять усе можливе, щоб допомогти вам знайти ідеальну пару.</p>
            </div>

            <div>
                <h2>
                    <strong className={styles.why_us}>
                        Чому ми? <img src="why-us.png" className={styles.whyUsIcon} alt="Why-us" />
                    </strong>
                </h2>
                <ul>
                    <li>Тренд моди: ми ретельно підбираємо кожну модель відповідно до останніх тенденцій.</li>
                    <li>Якість і комфорт: усі наші вироби виготовляються з високоякісних матеріалів з акцентом на комфорті.</li>
                    <li>Найвищий рівень обслуговування: наша мета — зробити ваші покупки приємними та неперевершеними.</li>
                </ul>
            </div>

            <div>
                <h2>
                    <strong className={styles.contact_us}>
                        Зв'яжіться з нами <img src="contact-us.png" className={styles.contactUsIcon} alt="Contact-us" />
                    </strong>
                </h2>
                <p className={styles.text}>Ми пишаємося тим, що створюємо не просто магазин, а спільноту любителів стилю. Приєднуйтесь до StyleStepHaven і діліться своїми враженнями з нами в соціальних мережах: [links to social networks].</p>
            </div>

            <p>Дякуємо, що обираєте StyleStepHaven. Ми з нетерпінням чекаємо на те, щоб ви крокували впевнено та стильно!</p>
        </div>
```

```
                </div>
                <div className={styles.about_us}>
                  <h2 className={styles.heading2}>Кількість відвідувачів</h2>
                  <form className={styles.form}>
                    <label className={styles.label}>
                      Кількість відвідувачів влітку:
                      <input type="number" name="summerVisitors" className={styles.input}
value={summerVisitors} onChange={(e) => setSummerVisitors(parseInt(e.target.value) || "")} />
                    </label>
                    <label className={styles.label}>
                      Кількість відвідувачів взимку:
                      <input type="number" name="winterVisitors" className={styles.input}
value={winterVisitors} onChange={(e) => setWinterVisitors(parseInt(e.target.value) || "")} />
                    </label>
                    <label className={styles.label}>
                      Кількість відвідувачів навесні:
                      <input type="number" name="springVisitors" className={styles.input}
value={springVisitors} onChange={(e) => setSpringVisitors(parseInt(e.target.value) || "")} />
                    </label>
                    <label className={styles.label}>
                      Кількість відвідувачів восени:
                      <input type="number" name="fallVisitors" className={styles.input}
value={fallVisitors} onChange={(e) => setFallVisitors(parseInt(e.target.value) || "")} />
                    </label>
                    <button type="button" className={styles.button} onClick={handleCalculate}>
                      Розрахувати відвідувачів
                    </button>
                  </form>
                </div>

                <div className={styles.about_us}>
                  <h2 className={styles.heading2}>Трафік і поведінка користувачів</h2>
                  <form className={styles.form}>
                    <label className={styles.label}>
                      Кількість унікальних відвідувачів:
                      <input type="number" name="uniqueVisitors" className={styles.input}
value={uniqueVisitors} onChange={(e) => setUniqueVisitors(parseInt(e.target.value) || "")} />
                    </label>
```

```
                    <label className={styles.label}>
                        Загальний час на сайті:
                        <input type="number" name="averageTimeOnSite" className={styles.input}
value={averageTimeOnSite} onChange={(e) => setAverageTimeOnSite(parseInt(e.target.value) || "")} />
                    </label>
                    <label className={styles.label}>
                        Відвідування однієї сторінки:
                        <input type="number" name="singlePageVisits" className={styles.input}
value={singlePageVisits} onChange={(e) => setSinglePageVisits(parseInt(e.target.value) || "")} />
                    </label>
                    <label className={styles.label}>
                        Загальна кількість відвідувань:
                        <input type="number" name="totalVisits" className={styles.input}
value={totalVisits} onChange={(e) => setTotalVisits(parseInt(e.target.value) || "")} />
                    </label>
                    <button type="button" className={styles.button}
onClick={handleTrafficCalculate}>
                        Розрахувати поведінку користувачів
                    </button>
                </form>
                {bounceRate !== "" && (
                    <div className={styles.result}>
                        <h3>Показник відмов: {bounceRate.toFixed(2)}%</h3>
                    </div>
                )}
            </div>

            <div className={styles.about_us}>
                <h2 className={styles.heading2}>Розподіл навантаження</h2>
                <form className={styles.form}>
                    <label className={styles.label}>
                        Кількість активних користувачів в моменти часу (через кому):
                        <input type="text" name="activeUsers" className={styles.input}
value={activeUsers.join(",")} onChange={(e) => setActiveUsers(e.target.value.split(",").map(Number))}
/>
                    </label>
                    <label className={styles.label}>
                        Навантаження від одного користувача:
```

```
                        <input type="number" name="userLoad" className={styles.input}
value={userLoad} onChange={(e) => setUserLoad(parseInt(e.target.value) || "")} />
                    </label>
                    <button type="button" className={styles.button} onClick={handleLoadCalculate}>
                        Розрахувати навантаження на сервер
                    </button>
                </form>
                {serverLoad !== "" && (
                    <div className={styles.result}>
                        <h3>Навантаження на сервер: {serverLoad.toFixed(2)}</h3>
                    </div>
                )}
            </div>

            <div className={styles.about_us}>
                <h2 className={styles.heading2}>Аналіз користувацької поведінки</h2>
                <form className={styles.form}>
                    <label className={styles.label}>
                        Матриця ймовірностей переходу (через кому, кожен рядок з нової лінії):
                        <textarea name="transitionMatrix" className={styles.input}
value={transitionMatrix.map((row) => row.join(",")).join("\n")} onChange={(e) =>
setTransitionMatrix(e.target.value.split("\n").map((row) => row.split(",").map(Number)))} />
                    </label>
                    <label className={styles.label}>
                        Поточний стан (індекс сторінки):
                        <input type="number" name="currentState" className={styles.input}
value={currentState} onChange={(e) => setCurrentState(parseInt(e.target.value) || "")} />
                    </label>
                    <button type="button" className={styles.button}
onClick={handleTransitionCalculate}>
                        Розрахувати ймовірності переходу
                    </button>
                </form>
                {nextStateProbabilities.length > 0 && (
                    <div className={styles.result}>
                        <h3>Ймовірності переходу до наступних станів:</h3>
                        <ul>
                            {nextStateProbabilities.map((prob, index) => (
```

```
                        <li key={index}>
                            До сторінки {index}: {prob.toFixed(2)}
                        </li>
                    ))}
                </ul>
            </div>
        )}
      </div>
    </div>
  );
};

export default About;
```

about.module.css

```
.text {
  font-size: 22px;
  font-style: italic;
}

.welcome {
  font-size: 5rem;
  text-transform: capitalize;
  text-shadow: 1px 1px 1px #000000, 2px 2px 1px #16ea1a;
  text-align: center;
}

.about {
  overflow: hidden;
}

.about_us {
  padding: 50px;
}

.historyBookIcon,
.teamIcon,
.whyUsIcon,
```

```
.contactUsIcon {
    width: 50px;
    height: 50px;
}

.history,
.mission,
.team,
.why_us,
.contact_us {
    font-size: 3rem;
    text-transform: capitalize;
    text-shadow: 1px 1px 1px #000000, 2px 2px 1px #16ea1a, 3px 3px 1px rgb(255, 0, 0);
    text-align: initial;
    border-radius: 100%;
}

/* styles.module.css */

.heading2 {
    font-size: 24px;
    font-weight: bold;
    margin-bottom: 16px;
}

.form {
    display: flex;
    flex-direction: column;
    gap: 16px;
}

.label {
    display: flex;
    flex-direction: column;
    font-size: 16px;
    margin-bottom: 8px;
}
```

```
.input {
    padding: 8px;
    font-size: 16px;
    margin-top: 4px;
    border: 1px solid #ccc;
    border-radius: 4px;
}

.button {
    padding: 12px;
    font-size: 16px;
    background-color: #27d547;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    transition: background-color 0.3s;
}

.button:hover {
    background-color: #259137;
}

.results {
    margin-top: 16px;
    padding: 16px;
    background-color: #f8f9fa;
    border: 1px solid #dee2e6;
    border-radius: 4px;
}

.results p {
    margin: 4px 0;
    font-size: 16px;
}
```

Basket.tsx

```
import React from "react";
```

```
import { useSelector, useDispatch } from "react-redux";
import ContactForm from "./forms/ContactForm";
import DeliveryForm from "./forms/DeliveryForm";
import { RootState } from "../../store/store";
import { cartActions } from "../../store/cart.slice";

interface BasketProps {}

const Basket: React.FC<BasketProps> = () => {
   const dispatch = useDispatch();
   const cartItems = useSelector((state: RootState) => state.cart.items);

   const handleIncrement = (id: number) => {
      dispatch(cartActions.add(cartItems.find((item) => item.id === id)!));
   };

   const handleDecrement = (id: number) => {
      dispatch(cartActions.decrease(id));
   };

   const handleRemove = (id: number) => {
      dispatch(cartActions.remove(id));
   };

   const handleDeliveryFormSubmit = () => {
      //  Логіка обробки даних форми
      console.log("Надіслані дані форми:", FormData);
   };

   return (
      <>
         <h1>Оформлення замовлення</h1>
         <div className="container">
            <div className="row">
               <div className="col">
                  <ContactForm />
               </div>
               <div className="col">
```

```
                        <DeliveryForm cartItems={cartItems} onSubmit={handleDeliveryFormSubmit}
onIncrement={handleIncrement} onDecrement={handleDecrement} onRemove={handleRemove} />
                    </div>
                </div>
            </div>
        </>
    );
};

export default Basket;
```

ContactForm.tsx

```
import React, { useState, ChangeEvent, FormEvent } from "react";
import styles from "../forms/ContactForm.module.css";

interface FormData {
    name: string;
    phone: string;
    city: string;
}

const ContactForm: React.FC = () => {
    const [formData, setFormData] = useState<FormData>({
        name: "",
        phone: "",
        city: "",
    });

    const handleChange = (e: ChangeEvent<HTMLInputElement>) => {
        const { name, value } = e.target;
        setFormData((prevFormData) => ({
            ...prevFormData,
            [name]: value,
        }));
    };

    const handleSubmit = (e: FormEvent<HTMLFormElement>) => {
        e.preventDefault();
```

```
        // Тут можна додати логіку для виправлення даних на сервері
        console.log("Відправка даних:", formData);
    };

    return (
        <form className={styles.contactForm} action="/submit_form" method="post"
onSubmit={handleSubmit}>
            <h1>1. Контактні дані</h1>
            <div>
                <label htmlFor="name">Ім'я:</label>
                <input className={styles.nameInput} type="text" id="name" name="name"
value={formData.name} onChange={handleChange} required />
            </div>
            <div>
                <label htmlFor="phone">Номер телефону:</label>
                <input className={styles.numberInput} type="tel" id="phone" name="phone"
value={formData.phone} onChange={handleChange} required />
            </div>
            <div>
                <label htmlFor="city">Місто:</label>
                <input className={styles.cityInput} type="text" id="city" name="city"
value={formData.city} onChange={handleChange} required />
            </div>
        </form>
    );
};
export default ContactForm;

ContactForm.module.css
.contactForm {
    border: 2px solid #14c923;
    border-radius: 7px;
    padding: 25px;
}

.nameInput {
    background: #191b1f;
    border-radius: 5px;
```

```
    margin: 10px 20px 0px 102px;
    border: 1px solid #ccc;
    color: #ffffff;
    font-size: 20px;
}

.numberInput {
    background: #191b1f;
    border-radius: 5px;
    margin: 10px 20px 0px 40px;
    border: 1px solid #ccc;
    color: #ffffff;
    font-size: 20px;
}

.cityInput {
    background: #191b1f;
    border-radius: 5px;
    margin: 10px 20px 0px 118px;
    border: 1px solid #ccc;
    color: #ffffff;
    font-size: 20px;
}

.button {
    border: 1px solid #ffffff;
    border-radius: 5px;
    padding: 7px 50px;
    background-color: rgb(14, 8, 8);
    color: #fff;
    font-family: Martian Mono, sans-serif;
    width: 100%;
    margin-top: 20px;
}

.button:hover {
    background-color: #16ea1a;
    color: black;
```

```
    border: 1px solid #111111;
}

DeliveryForm.tsx
import React, { useState, useEffect, ChangeEvent, FormEvent } from "react";
import styles from "../forms/DeliveryForm.module.css";
import { FaTrash } from "react-icons/fa";

interface CartItem {
    id: number;
    name: string;
    price: string;
    count: number;
}

interface DeliveryFormProps {
    cartItems: CartItem[];
    onSubmit: (formData: FormData) => void;
    onIncrement: (id: number) => void;
    onDecrement: (id: number) => void;
    onRemove: (id: number) => void;
}

interface FormData {
    order: CartItem[];
    amount: string;
    deliveryMethod: string;
    paymentMethod: string;
}

const DeliveryForm: React.FC<DeliveryFormProps> = ({ cartItems, onIncrement, onDecrement,
onRemove }) => {
    const [formData, setFormData] = useState<FormData>({
        order: cartItems || [],
        amount: "",
        deliveryMethod: "courier",
        paymentMethod: "creditCard",
    });
```

```
useEffect(() => {
  setFormData((prevFormData) => ({
    ...prevFormData,
    order: cartItems || [],
  }));
}, [cartItems]);

const handleChange = (e: ChangeEvent<HTMLSelectElement | HTMLInputElement>) => {
  const { name, value } = e.target;
  setFormData({
    ...formData,
    [name]: value,
  });
};

const handleSubmit = async (e: FormEvent<HTMLFormElement>) => {
  e.preventDefault();

  try {
    const response = await fetch("http://127.0.0.1:8000/api/submit-order/", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify(formData),
    });

    if (response.ok) {
      const responseData = await response.json();

      // Перевірка наличия URL-адреси властивостей в об'єкті
      if (responseData && responseData.url) {
        const url = responseData.url;
        console.log("Заказ успішно відправлений.");
        window.location.assign(url);
      } else {
        console.error("Відповідь сервера не містить властивості 'url'");
```

```
                }
            }
            {
                console.error("Не вдалося відправити замовлення");
            }
        } catch (error) {
            console.error("Помилка при відправці замовлення:", error);
        }
    };

    const calculateTotalPrice = () => {
        if (cartItems && cartItems.length > 0) {
            const totalPrice = cartItems.reduce((total, item) => total + parseFloat(item.price) *
item.count, 0);
            return ` ${totalPrice.toFixed(2)} ₴`;
        }
        return "0.00 ₴";
    };

    const handleIncrement = (id: number) => {
        onIncrement(id);
    };

    const handleDecrement = (id: number) => {
        onDecrement(id);
    };

    const handleRemove = (id: number) => {
        onRemove(id);
    };

    return (
        <form className={styles.deliveryForm} action="/submit_order" method="post"
onSubmit={handleSubmit}>
            <h1>2. Виберіть варіанти доставки</h1>
            <div>
                <label htmlFor="order">Замовлення:</label>
                <ul>
```

```
                {cartItems && cartItems.length > 0 ? (
                  cartItems.map((item) => (
                    <li key={item.id}>
                      {item.name} - Кількість: {item.count}; Ціна - {item.price} ₴<br></br>
                      <button className={styles.increment} type="button" onClick={() =>
handleIncrement(item.id)}>
                        ✚
                      </button>
                      <button className={styles.decrement} type="button" onClick={() =>
handleDecrement(item.id)}>
                        ▬
                      </button>
                      <button className={styles.deleteButton} type="button" onClick={() =>
handleRemove(item.id)}>
                        <FaTrash className={styles.deleteIcon} />
                      </button>
                    </li>
                  ))
                ) : (
                  <li>У замовленні немає товарів</li>
                )}
              </ul>
            </div>

            <div>
              <label htmlFor="totalPrice">Загальна сума:</label>
              <span>{calculateTotalPrice()}</span>
            </div>

            <div>
              <label htmlFor="deliveryMethod">Метод доставки:</label>
              <select className={styles.deliveryInput} id="deliveryMethod"
name="deliveryMethod" value={formData.deliveryMethod} onChange={handleChange} required>
                <option value="courier">Кур'єр</option>
              </select>
            </div>

            <div>
```

```
                    <label htmlFor="paymentMethod">Спосіб оплати:</label>
                    <select className={styles.paymentMethod} id="paymentMethod"
name="paymentMethod" value={formData.paymentMethod} onChange={handleChange} required>
                        <option value="creditCard">Кредитна карта</option>
                        <option value="paypal">PayPal</option>
                    </select>
                </div>

                <button className={styles.button} type="submit">
                    Надіслати
                </button>
            </form>
        );
    };

    export default DeliveryForm;

    DeliveryForm.module.css
    .deliveryForm {
        border: 2px solid #14c923;
        border-radius: 7px;
        padding: 25px;
    }

    .sum {
        background: #191b1f;
        border-radius: 5px;
        margin: 10px 20px 0px 102px;
        border: 1px solid #ccc;
        color: #ffffff;
        font-size: 20px;
    }

    .deliveryInput {
        background: #191b1f;
        border-radius: 5px;
        margin: 10px 20px 0px 25px;
        border: 1px solid #ccc;
```

```
    color: #ffffff;
    font-size: 20px;
}

.paymentMethod {
    background: #191b1f;
    border-radius: 5px;
    margin: 10px 20px 0px 21px;
    border: 1px solid #ccc;
    color: #ffffff;
    font-size: 20px;
}

.button {
    border: 1px solid #ffffff;
    border-radius: 5px;
    padding: 7px 50px;
    background-color: rgb(14, 8, 8);
    color: #fff;
    font-family: Martian Mono, sans-serif;
    width: 100%;
    margin-top: 20px;
}

.button:hover {
    background-color: #16ea1a;
    color: black;
    border: 1px solid #111111;
}

.decrement,
.increment {
    padding: 2px 3px 3px 3px;
    margin: 5px;
    width: 30px;
    border-radius: 10px;
    border: none;
    background-color: #797979;
```

```
}

.decrement:hover,
.increment:hover {
    background-color: #c4c6c1;
    box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);
    transition: 100ms;
}

.deleteIcon {
    color: #ca5252;
    float: right;
    top: -50px;
    cursor: pointer;
    transition: color, transform 500ms ease;
}

.deleteIcon:hover {
    color: rgb(207, 7, 7);
    transform: scale(1.5);
}

.deleteButton {
    background-color: #111111;
    border: none;
}
```

BuyButton.tsx

```
import React from "react";
import styles from "./BuyButton.module.css";
import { useDispatch } from "react-redux";
import { cartActions } from "../../store/cart.slice";
import { motion } from "framer-motion";
import { ProductData } from "../product/Product";

interface BuyButtonProps {
    product: ProductData;
}
```

```
const BuyButton: React.FC<BuyButtonProps> = ({ product }) => {
  const dispatch = useDispatch();

  const handleBuyClick = (event: React.MouseEvent<HTMLButtonElement, MouseEvent>) =>
{
    event.preventDefault();
    dispatch(cartActions.add(product));
  };

  return (
    <div className="example-container">
      <motion.button whileHover={{ scale: 1.2 }} whileTap={{ scale: 0.8 }}
className={styles.buyButton} onClick={handleBuyClick}>
        Купити
      </motion.button>
    </div>
  );
};

export default BuyButton;
```

BuyButton.module.css

```
.buyButton {
  border: 1px solid #ffffff;
  border-radius: 5px;
  padding: 7px 50px;
  background-color: rgb(14, 8, 8);
  color: #fff;
  font-family: Martian Mono, sans-serif;

  overflow: hidden;
}

.buyButton:hover {
  background-color: #16ea1a;
  color: black;
```

```
    border: 1px solid #111111;
    transform: scale(1.1);
}

.buyButton:before {
    font-family: FontAwesome;
    content: "\f07a";
    position: absolute;
    top: 11px;
    left: -30px;
    transition: all 200ms ease;
}
```

Women.tsx

```
import React from "react";
import Product from "../product/Product";

const Women: React.FC = () => {
    return (
        <div>
            <h2>Жіночі товари</h2>
            <Product filter="women" />
        </div>
    );
};

export default Women;
```

Men.tsx

```
import Product from "../product/Product";

const Men: React.FC = () => {
    return (
        <div>
            <h2>Чоловічі товари</h2>
            <Product filter="men" />
        </div>
    );
```

```
};

export default Men;
```

Kids.tsx

```
import Product from "../product/Product";

const Kids: React.FC = () => {
  return (
    <div>
      <h3>Товари для дітей</h3>
      <Product filter="kids" />
    </div>
  );
};

export default Kids;
```

Contacts.tsx

```
import React from "react";
import styles from "../contacts/Contacts.module.css";

const Contacts: React.FC = () => {
  return (
    <div className={styles.container}>
      <h1>Контакти</h1>
      <div>
        <h2 className={styles.heading2}>Зв'яжіться з нами</h2>
        <form className={styles.form}>
          <label className={styles.label}>
            Ім'я:
            <input type="text" name="name" className={styles.input} />
          </label>
          <label className={styles.label}>
            Email:
            <input type="email" name="email" className={styles.input} />
          </label>
          <label className={styles.label}>
```

```
                Повідомлення:
                <textarea name="message" className={styles.textarea}></textarea>
              </label>
              <button type="submit" className={styles.button}>
                Відправити
              </button>
            </form>
          </div>
          <div className={styles.companyInfo}>
            <h2 className={styles.heading2}>Інформація про компанію</h2>
            <p>Назва компанії: ТОВ "StyleStepHaven"</p>
            <p>Адреса: м. Київ, вул. Хрещатик, 1</p>
            <p>Телефон: +380 44 123 4567</p>
            <p>Email: info@market.com</p>
          </div>
          <div className={styles.map}>
            <h2 className={styles.heading2}>Наше розташування</h2>
            <iframe
src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d25392.495018634047!2d30.498845
699377625!3d50.44923464720906!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x40d4cf84f
27d59a7%3A0x63d4651b1426b598!2z0JrQvtC80L_RgNCw0LvRjNC90YvQuSDQv9C-
0YDRi9C60LjQuSDQvdCw0YHRgtGA0L7QutCw!5e0!3m2!1suk!2sua!4v1587642603135!5m2!1suk!2
sua" width="600" height="450" style={{ border: 0 }} allowFullScreen={false}
loading="lazy"></iframe>
          </div>
          <div className={styles.social}>
            <h2 className={styles.heading2}>Ми в соціальних мережах</h2>
            <p>
              <a href="https://www.facebook.com" className={styles.link}>
                Facebook
              </a>
            </p>
            <p>
              <a href="https://www.twitter.com" className={styles.link}>
                Twitter
              </a>
            </p>
            <p>
```

```
                <a href="https://www.instagram.com" className={styles.link}>
                  Instagram
                </a>
              </p>
            </div>
          </div>
    );
};

export default Contacts;
```

Contacts.module.css

```
/* Contacts.module.css */
.container {
  max-width: 800px;
  margin: 0 auto;
  padding: 20px;
  font-family: Arial, sans-serif;
}

.heading2 {
  color: #555;
}

.form {
  display: flex;
  flex-direction: column;
  gap: 10px;
}

.label {
  display: flex;
  flex-direction: column;
  font-size: 14px;
  color: #444;
}

.input,
```

```
.textarea {
    padding: 8px;
    font-size: 14px;
    border: 1px solid #ccc;
    border-radius: 4px;
}

.button {
    padding: 10px 15px;
    font-size: 16px;
    color: white;
    background-color: #219e32;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    align-self: flex-start;
}

.button:hover {
    background-color: #158830;
}

.companyInfo,
.map,
.social {
    margin-top: 20px;
}

.link {
    color: #26802b;
    text-decoration: none;
}

.link:hover {
    text-decoration: underline;
}
```

Filter.tsx

```
import { useState, FC } from "react";
import styles from "./Filter.module.css";

interface FilterProps {
  onFilterChange: (category: string) => void;
}

const Filter: FC<FilterProps> = ({ onFilterChange }) => {
  const [activeFilter, setActiveFilter] = useState("all");

  const handleButtonClick = (category: string) => {
    setActiveFilter(category);
    onFilterChange(category);
  };

  return (
    <div className={styles.filterContainer}>
      <button className={`${styles.button} ${activeFilter === "all" ? styles.activeButton :
""}`} onClick={() => handleButtonClick("all")}>
        Усі
      </button>
      <button className={`${styles.button} ${activeFilter === "men" ? styles.activeButton :
""}`} onClick={() => handleButtonClick("men")}>
        Чоловічі
      </button>
      <button className={`${styles.button} ${activeFilter === "women" ? styles.activeButton
: ""}`} onClick={() => handleButtonClick("women")}>
        Жіночі
      </button>
      <button className={`${styles.button} ${activeFilter === "kids" ? styles.activeButton :
""}`} onClick={() => handleButtonClick("kids")}>
        Дитячі
      </button>
    </div>
  );
};

export default Filter;
```

Filter.module.css

```
/* Filter.module.css */
.filterContainer {
    display: flex;
    justify-content: center;
    margin-bottom: 20px;
}

.button {
    background-color: #282928;
    color: white;
    border: none;
    padding: 10px 20px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    cursor: pointer;
    border-radius: 4px;
    transition: background-color 0.3s;
    margin: 5px;
}

.button:last-child {
    margin-right: 0; /* У последней кнопки нет правого отступа */
}

.button:hover,
.button:active,
.button:focus {
    background-color: #45a049; /* Цвет для наведения и активного состояния */
}

.button:focus {
    outline: none; /* Убирает стандартный браузерный контур фокуса */
}
```

```
.activeButton {
  background-color: #45a049; /* Цвет для активной кнопки */
}
```

Footer.tsx

```
import { FC } from "react";
import styles from "./Footer.module.css";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faFacebook, faInstagram, faTelegram, faYoutube } from "@fortawesome/free-brands-svg-icons";
import { faPhone, faEnvelope } from "@fortawesome/free-solid-svg-icons";

const Footer: FC = () => {
  return (
    <footer className={styles.footer}>
      <div className="container text-center">
        <div className="row">
          <div className="col">
            <span className={styles.footerName}>НАЗВА КОМПАНІЇ</span>
            <p className={styles.footerItem}>StyleStepHaven — провідна взуттєва компанія, що спеціалізується на розробці, виробництві та продажу стильного та зручного взуття.</p>
          </div>
          <div className="col">
            <span className={styles.footerName}>ПРОДУКЦІЯ</span>
            <p className={styles.footerItem}></p>
          </div>
          <div className="col">
            <span className={styles.footerName}>КОРИСНІ ПОСИЛАННЯ</span>
            <div className={styles.footerItem}>
              <ul>
                <a href="#">
                  <li>Події</li>
                </a>
                <a href="#">
                  <li>Команда</li>
                </a>
                <a href="#">
```

```
                        <li>Галерея</li>
                    </a>
                </ul>
            </div>
        </div>
        <div className="col">
            <span className={styles.footerName}>КОНТАКТИ</span>
            <div className={styles.footerItem}>
                <div className="contact">
                    <span>
                        <FontAwesomeIcon icon={faPhone} />
                          123-456-789
                    </span>
                    <br />
                    <span>
                        <FontAwesomeIcon icon={faEnvelope} />
                          info@market.com
                    </span>
                </div>
                <div>
                    <a href="#">
                        <FontAwesomeIcon className={styles.socials} icon={faFacebook} />
                    </a>
                    <a href="#">
                        <FontAwesomeIcon className={styles.socials} icon={faInstagram} />
                    </a>
                    <a href="#">
                        <FontAwesomeIcon className={styles.socials} icon={faTelegram} />
                    </a>
                    <a href="#">
                        <FontAwesomeIcon className={styles.socials} icon={faYoutube} />
                    </a>
                </div>
            </div>
        </div>
    </div>
</footer>
```

```
    );
};

export default Footer;
```

Footer.module.css

```
a {
    text-decoration: none;
    color: #ffffff;
}
a:hover {
    color: rgb(26, 242, 7);
}

.footer {
    background-color: #000000;
    color: #ffffff;
    font-family: Martian Mono, sans-serif;
    width: 100%;
    margin-top: auto;
}

.footerName {
    text-decoration: underline rgb(26, 242, 7);
}

.footerItem {
    padding: 10px;
    text-align: left;
}

.socials {
    font-size: 25px;
    padding: 10px;
}
```

HeaderAuth.tsx

```
// HeaderAuth.tsx
```

```
import React, { useState, useCallback, useEffect } from "react";
import { NavLink } from "react-router-dom";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faShoppingCart } from "@fortawesome/free-solid-svg-icons";
import { FaTrash } from "react-icons/fa";
import styles from "./Header.module.css";
import logo from "/public/logo.png";
import Search from "../search/Search";
import { useSelector, useDispatch } from "react-redux";
import { RootState } from "../../store/store";
import { cartActions } from "../../store/cart.slice";

interface HeaderAuthProps {
  isLoggedIn: boolean;
}

const HeaderAuth: React.FC<HeaderAuthProps> = ({ isLoggedIn }) => {
  const [isActive, setIsActive] = useState(false);
  const dispatch = useDispatch();

  const toggleActive = () => {
    setIsActive((prev) => !prev);
  };

  const cartItems = useSelector((state: RootState) => state.cart.items);

  const handleRemoveProduct = useCallback(
    (id: number) => {
      dispatch(cartActions.remove(id));
    },
    [dispatch]
  );

  const handleIncrement = useCallback(
    (id: number) => {
      dispatch(cartActions.add(cartItems.find((item) => item.id === id)!));
    },
    [dispatch, cartItems]
```

```
    );

    const handleDecrement = useCallback(
      (id: number) => {
        dispatch(cartActions.decrease(id));
      },
      [dispatch]
    );

    useEffect(() => {
      const totalItemsCount = cartItems.reduce((total, item) => total + item.count, 0);
      console.log(totalItemsCount);
    }, [cartItems]);

    return (
      <header className={styles.header}>
        <img src={logo} alt="logo" />

        <Search />
        <nav className={styles.navigation}>
          <NavLink to="#" className={`${styles.navitem} ${isActive ? styles.active : ""}`}
onClick={toggleActive}>
            <FontAwesomeIcon icon={faShoppingCart} />
            {cartItems.length > 0 && <span
className={styles.cartCount}>{cartItems.reduce((total, item) => total + item.count, 0)}</span>}
          </NavLink>

          <NavLink className={styles.navitem} to="/">
            Головна
          </NavLink>
          <NavLink className={styles.navitem} to="/about">
            Про нас
          </NavLink>
          <NavLink className={styles.navitem} to="/contacts">
            Контакти
          </NavLink>
          {isLoggedIn && (
            <NavLink className={styles.navitem} to="/profile">
```

```
                Профіль
              </NavLink>
            )}
            {isActive && (
              <div>
                <div>
                  {cartItems && cartItems.length > 0 ? (
                    <div className={styles.shop}>
                      {cartItems.map((item) => (
                        <div className={styles.shopCart} key={item.id}>
                          <div className="row ">
                            <div className="col-3">{item.image && <img
className={styles.img} src={item.image} alt={item.name} />}</div>
                            <div className="col-8">
                              <p className={styles.name}>Name: {item.name}</p>
                              <p className={styles.price}>Price: {item.price} ₴</p>
                            </div>
                            <div className="col">
                              <FaTrash className={styles.deleteIcon} onClick={() =>
handleRemoveProduct(item.id)} />
                            </div>
                          </div>

                          <div className="row">
                            <div className="col">
                              <div className={styles.quantity}>
                                <button className={styles.increment} onClick={() =>
handleDecrement(item.id)}>
                                  −
                                </button>
                                <span>{item.count}</span>
                                <button className={styles.decrement} onClick={() =>
handleIncrement(item.id)}>
                                  +
                                </button>
                              </div>
                            </div>
                            <div className="col">
```

```
                                    <NavLink to="/basket" className={styles.cartButton}>
                                        <span className={styles.cart}>Перейти в кошик</span>
                                    </NavLink>
                                </div>
                            </div>
                        </div>
                    ))}
                </div>
            ) : (
                <div className={styles.shop}>
                    <span>Немає товарів</span>
                </div>
            )}
        </div>
    </div>
  )}
 </nav>
</header>
  );
};
export default HeaderAuth;
```

HeaderAuth.module.css

```
* {
  margin: 0;
  box-sizing: border-box;
}

body {
  padding-top: 85px;
}

.header {
  top: 0;
  left: 0;
  width: 100%;
  position: fixed;
  display: flex;
  justify-content: space-between;
```

```
    align-items: center;
    background-color: #000000;
    color: #fff;
    padding: 10px 20px;
    z-index: 1000;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.navitem,
.login {
    font-family: Martian Mono, sans-serif;
    background-color: #000000;
    color: #fff;
    border: none;
    padding: 10px 20px;
    font-size: 16px;
    cursor: pointer;
    border-radius: 10px;
    text-decoration: none;
}

.navitem:hover {
    background-color: rgb(234, 222, 122);
    color: #000000;
}

.login:hover {
    background-color: #16ea1a;
    color: #000000;
}

@media (max-width: 768px) {
    .header {
        flex-direction: column;
        align-items: center;
    }

    .navitem,
```

```
    .login {
        margin-top: 10px;
    }
}

.active {
    background-color: rgb(234, 222, 122);
    color: #000000;
}

header .shop {
    justify-content: center;
    position: absolute;
    top: 30px;
    right: 500px;
    width: 450px;
    max-height: 500px;
    overflow-y: auto;
    background: #222b31;
    border-radius: 10px;
    z-index: 999;
}

header .shop::-webkit-scrollbar {
    width: 10px;
}

header .shop::-webkit-scrollbar-thumb {
    background-color: #16ea1a;
    border-radius: 6px;
}

header .shop::-webkit-scrollbar-track {
    background-color: #3a4750;
}

.img {
    width: 100px;
```

```
    height: 90px;
    float: left;
    margin-right: 20px;
    border-radius: 5px;
}

.name {
    font-size: 20px;
    margin-bottom: 10px;
}

.price {
    color: #797979;
    font-weight: 600;
}

.shopCart {
    padding: 20px;
    padding-bottom: 0;
    border-bottom: 1px solid #3a4750; /* Add a border between items */
}

.deleteIcon {
    color: #ca5252;
    float: right;
    position: relative;
    top: -50px;
    cursor: pointer;
    transition: color, transform 500ms ease;
}

.deleteIcon:hover {
    color: rgb(207, 7, 7);
    transform: scale(1.5);
}

.cartCount {
    position: absolute;
```

```
    top: 15px;
    right: 1;
    background-color: red;
    color: white;
    border-radius: 50%;
    padding: 4px 8px;
    font-size: 10px;
}

.quantity {
    color: #16ea1a;
    font-weight: bold;
}

.decrement,
.increment {
    padding: 2px 3px 3px 3px;
    margin: 5px;
    width: 30px;
    border-radius: 10px;
    border: none;
    background-color: #797979;
}

.decrement:hover,
.increment:hover {
    background-color: #c4c6c1;
    box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);
    transition: 100ms;
}

.cartButton {
    width: 120px;
    margin-bottom: 10px;
    text-decoration: none;
    color: #fff;
    background-color: #1e7332;
    padding: 10px 20px;
```

```
    border-radius: 5px;
    margin-left: 10px; /* Adjust the margin as needed */
    display: flex;
    transition: background-color 0.3s ease;
}

.cartButton:hover {
    background-color: #3ab85e;
    transition: background-color 0.3s ease;
    color: #000000;
}

.cartIcon {
    font-size: 20px;
}
```

Header.tsx

```
// Header.tsx
import React, { useState, useCallback, useEffect } from "react";
import { NavLink } from "react-router-dom";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faShoppingCart } from "@fortawesome/free-solid-svg-icons";
import { FaTrash } from "react-icons/fa";
import styles from "./Header.module.css";
import logo from "/public/logo.png";
import Search from "../search/Search";
import { useSelector, useDispatch } from "react-redux";
import { RootState } from "../../store/store";
import { cartActions } from "../../store/cart.slice";

// Оновлюємо інтерфейс HeaderProps
interface HeaderProps {
    isLoggedIn: boolean; // Додаємо проп isLoggedIn
}

const Header: React.FC<HeaderProps> = ({ isLoggedIn }) => {
    // Передаємо пропс isLoggedIn
    const [isActive, setIsActive] = useState(false);
```

```
const dispatch = useDispatch();

const toggleActive = () => {
  setIsActive((prev) => !prev);
};

const cartItems = useSelector((state: RootState) => state.cart.items);

const handleRemoveProduct = useCallback(
  (id: number) => {
    dispatch(cartActions.remove(id));
  },
  [dispatch]
);

const handleIncrement = useCallback(
  (id: number) => {
    dispatch(cartActions.add(cartItems.find((item) => item.id === id)!));
  },
  [dispatch, cartItems]
);

const handleDecrement = useCallback(
  (id: number) => {
    dispatch(cartActions.decrease(id));
  },
  [dispatch]
);

useEffect(() => {
  const totalItemsCount = cartItems.reduce((total, item) => total + item.count, 0);
  console.log(totalItemsCount);
}, [cartItems]);

return (
  <header className={styles.header}>
    <img src={logo} alt="логотип" />
```

```
                <Search />
                <nav className={styles.navigation}>
                  <NavLink to="#" className={`${styles.navitem} ${isActive ? styles.active : ""}`}
onClick={toggleActive}>
                    <FontAwesomeIcon icon={faShoppingCart} />
                    {cartItems.length > 0 && <span
className={styles.cartCount}>{cartItems.reduce((total, item) => total + item.count, 0)}</span>}
                  </NavLink>

                  <NavLink className={styles.navitem} to="/">
                    Головна
                  </NavLink>
                  <NavLink className={styles.navitem} to="/about">
                    Про нас
                  </NavLink>
                  <NavLink className={styles.navitem} to="/contacts">
                    Контакти
                  </NavLink>
                  {/* Залежно від isLoggedIn показуємо різні посилання */}
                  {isLoggedIn ? (
                    <NavLink className={styles.navitem} to="/profile">
                      Профіль
                    </NavLink>
                  ) : (
                    <NavLink className={styles.login} to="/login">
                      Увійти
                    </NavLink>
                  )}
                  {isActive && (
                    <div>
                      <div>
                        {cartItems && cartItems.length > 0 ? (
                          <div className={styles.shop}>
                            {cartItems.map((item) => (
                              <div className={styles.shopCart} key={item.id}>
                                <div className="row ">
                                  <div className="col-3">{item.image && <img
className={styles.img} src={item.image} alt={item.name} />}</div>
```

```
                                    <div className="col-8">
                                        <p className={styles.name}>Назва: {item.name}</p>
                                        <p className={styles.price}>Ціна: {(Number(item.price) *
Number(item.count)).toFixed(2)} ₴</p>
                                    </div>
                                    <div className="col">
                                        <FaTrash className={styles.deleteIcon} onClick={() =>
handleRemoveProduct(item.id)} />
                                    </div>
                                </div>

                                <div className="row">
                                    <div className="col">
                                        <div className={styles.quantity}>
                                            <button className={styles.decrement} onClick={() =>
handleDecrement(item.id)}>
                                                −
                                            </button>
                                            <span>{item.count}</span>
                                            <button className={styles.increment} onClick={() =>
handleIncrement(item.id)}>
                                                +
                                            </button>
                                        </div>
                                    </div>
                                    <div className="col">
                                        <NavLink to="/basket" className={styles.cartButton}>
                                            <span className={styles.cart}>Перейти до кошика</span>
                                        </NavLink>
                                    </div>
                                </div>
                            </div>
                        ))}
                    </div>
                ) : (
                    <div className={styles.shop}>
                        <span>Немає товарів</span>
                    </div>
```

```
                    )}
                  </div>
                </div>
              )}
            </nav>
          </header>
    );
};

export default Header;
```

Header.module.css

```
* {
    margin: 0;
    box-sizing: border-box;
}

body {
    padding-top: 85px;
}

.header {
    top: 0;
    left: 0;
    width: 100%;
    position: fixed;
    display: flex;
    justify-content: space-between;
    align-items: center;
    background-color: #000000;
    color: #fff;
    padding: 10px 20px;
    z-index: 1000;
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

.navitem,
.login {
```

```
    font-family: Martian Mono, sans-serif;
    background-color: #000000;
    color: #fff;
    border: none;
    padding: 10px 20px;
    font-size: 16px;
    cursor: pointer;
    border-radius: 10px;
    text-decoration: none;
}

.navitem:hover {
    background-color: rgb(234, 222, 122);
    color: #000000;
}

.login:hover {
    background-color: #16ea1a;
    color: #000000;
}

@media (max-width: 768px) {
    .header {
        flex-direction: column;
        align-items: center;
    }

    .navitem,
    .login {
        margin-top: 10px;
    }
}

.active {
    background-color: rgb(234, 222, 122);
    color: #000000;
}
```

```
header .shop {
    justify-content: center;
    position: absolute;
    top: 30px;
    right: 500px;
    width: 450px;
    max-height: 500px;
    overflow-y: auto;
    background: #222b31;
    border-radius: 10px;
    z-index: 999;
}

header .shop::-webkit-scrollbar {
    width: 10px;
}

header .shop::-webkit-scrollbar-thumb {
    background-color: #16ea1a;
    border-radius: 6px;
}

header .shop::-webkit-scrollbar-track {
    background-color: #3a4750;
}

.img {
    width: 100px;
    height: 90px;
    float: left;
    margin-right: 20px;
    border-radius: 5px;
}

.name {
    font-size: 20px;
    margin-bottom: 10px;
}
```

```
.price {
    color: #797979;
    font-weight: 600;
}

.shopCart {
    padding: 20px;
    padding-bottom: 0;
    border-bottom: 1px solid #3a4750;
}

.deleteIcon {
    color: #ca5252;
    float: right;
    position: relative;
    top: -50px;
    cursor: pointer;
    transition: color, transform 500ms ease;
}

.deleteIcon:hover {
    color: rgb(207, 7, 7);
    transform: scale(1.5);
}

.cartCount {
    position: absolute;
    top: 15px;
    right: 1;
    background-color: red;
    color: white;
    border-radius: 50%;
    padding: 4px 8px;
    font-size: 10px;
}

.quantity {
```

```
    color: #16ea1a;
    font-weight: bold;
}

.decrement,
.increment {
    padding: 2px 3px 3px 3px;
    margin: 5px;
    width: 30px;
    border-radius: 10px;
    border: none;
    background-color: #797979;
}

.decrement:hover,
.increment:hover {
    background-color: #c4c6c1;
    box-shadow: 0 0 5px rgba(0, 0, 0, 0.3);
    transition: 100ms;
}

.cartButton {
    width: 120px;
    margin-bottom: 10px;
    text-decoration: none;
    color: #fff;
    background-color: #1e7332;
    padding: 10px 20px;
    border-radius: 5px;
    margin-left: 10px;
    display: flex;
    transition: background-color 0.3s ease;
}

.cartButton:hover {
    background-color: #3ab85e;
    transition: background-color 0.3s ease;
    color: #000000;
```

```
}

.cartIcon {
  font-size: 20px;
}
```

Home.tsx

```
import React, { useEffect, useState } from "react";
import { useDispatch, useSelector } from "react-redux";
import { RootState } from "../../store/store";
import Product from "../product/Product";
import ShoeCategories from "../shoeCategories/ShoeCategories";
import styles from "./Home.module.css";
import Filter from "../filter/Filter";
import { updateSearchDuration } from "../../store/search.slice";

interface HomeProps {}

const Home: React.FC<HomeProps> = () => {
  const [filter, setFilter] = useState<string>("all");
  const searchDuration = useSelector((state: RootState) => state.search.searchDuration);
  const dispatch = useDispatch();

  const handleFilterChange = (category: string) => {
    setFilter(category);
  };

  useEffect(() => {
    window.scrollTo(0, 0); // Прокручування сторінки вгору під час монтажу компонента
    const startTime = performance.now();

    setTimeout(() => {
      const endTime = performance.now();
      dispatch(updateSearchDuration(endTime - startTime));
    }, 10);
  }, [dispatch]); // Порожній масив залежностей, щоб код викликався лише один раз

  return (
```

```
      <div className={styles.home}>
        <br></br>
        <p>Час завантаження: {searchDuration.toFixed(3)} мс</p>
        <ShoeCategories />
        <Filter onFilterChange={handleFilterChange} />
        <Product filter={filter} />
      </div>
    );
};

export default Home;
```

Registration.tsx

```
import React, { useState, ChangeEvent, FormEvent } from "react";
import { Link } from "react-router-dom";
import axios from "axios";
import { Modal, Button } from "@mui/material";
import styles from "./Registration.module.css";

interface FormData {
  first_name: string;
  last_name: string;
  username: string;
  password: string;
}

const Registration: React.FC = () => {
  const [formData, setFormData] = useState<FormData>({
    first_name: "",
    last_name: "",
    username: "",
    password: "",
  });
  const [showModal, setShowModal] = useState(false);

  const handleChange = (e: ChangeEvent<HTMLInputElement>) => {
    const { name, value } = e.target;
```

```
        setFormData({
          ...formData,
          [name]: value,
        });
      };

      const handleSubmit = async (e: FormEvent<HTMLFormElement>) => {
        e.preventDefault();
        try {
          const response = await axios.post("http://localhost:8000/api/user/create/", formData);
          const token = response.data.token; // Передбачається, що сервер повертає токен у форматі JSON
          localStorage.setItem("access_token", token); // Збереження токена в localStorage
          setShowModal(true); // Показати модальне вікно під час успішної реєстрації
        } catch (error) {
          console.error("Помилка під час реєстрації:", error);
        }
      };

      const closeModal = () => {
        setShowModal(false);
      };

      return (
        <div className={styles.container}>
          <form className={styles.regForm} onSubmit={handleSubmit}>
            <div className={styles.name}>
              <label htmlFor="firstName">Ім'я:</label>
              <input className={styles.nameInput} type="text" id="firstName" name="first_name" value={formData.first_name} onChange={handleChange} required />
            </div>
            <div className={styles.surname}>
              <label htmlFor="lastName">Прізвище:</label>
              <input className={styles.surnameInput} type="text" id="lastName" name="last_name" value={formData.last_name} onChange={handleChange} required />
            </div>
            <div className={styles.username}>
              <label htmlFor="username">Логін:</label>
```

```
                    <input className={styles.usernameInput} type="text" id="username"
name="username" value={formData.username} onChange={handleChange} required />
                </div>
                <div className={styles.password}>
                    <label htmlFor="password">Пароль:</label>
                    <input className={styles.passwordInput} type="password" id="password"
name="password" value={formData.password} onChange={handleChange} required />
                </div>
                <button className={styles.buttonRegistration} type="submit">
                    Реєстрація
                </button>
                <Link to="/login" className={styles.buttonLogin}>
                    Логін
                </Link>
            </form>
            <Modal open={showModal} onClose={closeModal}>
                <div className={styles.modal}>
                    <div className={styles.modalContent}>
                        <h2>Реєстрація успішна!</h2>
                        <Button onClick={closeModal} variant="contained">
                            Закрити
                        </Button>
                    </div>
                </div>
            </Modal>
        </div>
    );
};

export default Registration;
```

Registration.module.css

```
/* Registration.module.css */

.container {
    display: flex;
    align-items: center;
    justify-content: center;
```

```
    height: 100vh;
}

.regForm {
    text-align: center;
    padding: 25px;
    margin: 10px;
    border: 5px solid #16ea1a;
    border-radius: 10px;
    font-family: Kanit, sans-serif;
    width: 300px;
}

.username,
.password,
.name,
.surname {
    padding: 10px 10px;
}

.usernameInput,
.passwordInput,
.nameInput,
.surnameInput {
    background: #191b1f;
    border-radius: 5px;
    margin: 10px 0;
    border: 1px solid #ccc;
    color: #ffffff;
    font-size: 16px;
    padding: 8px;
    width: 100%;
    box-sizing: border-box;
}

.usernameInput {
    margin: 10px 0;
}
```

```
.surnameInput {
    margin: 10px 0;
}

.passwordInput {
    margin-top: 10px;
}

.usernameInput:focus,
.passwordInput:focus,
.nameInput:focus,
.surnameInput:focus {
    outline: none;
    border-color: #16ea1a;
}

.buttonRegistration,
.buttonLogin {
    border: 1px solid #ffffff;
    border-radius: 5px;
    padding: 8px 15px;
    margin: 10px;
    background-color: #191b1f;
    color: #ffffff;
    font-size: 16px;
    transition: 0.35s;
    cursor: pointer;
}

.buttonLogin:hover {
    background-color: rgb(234, 222, 122);
    color: #191b1f;
    border: 1px solid #111111;
}

.buttonRegistration:hover {
    background-color: #16ea1a;
```

```
    color: #191b1f;
    border: 1px solid #111111;
}

.modal {
    position: fixed;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    background-color: rgb(51, 50, 50);
    border-radius: 8px;
    padding: 20px;
    max-width: 400px;
    width: 100%;
}

.modalContent {
    text-align: center;
}

.modalButton {
    margin-top: 20px;
}
```

Login.tsx

```
import { useEffect, useState, ChangeEvent, FormEvent } from "react";
import { Link, useNavigate } from "react-router-dom";
import { useDispatch } from "react-redux";
import axios from "axios";
import { login } from "../../store/auth.slice";
import styles from "./Login.module.css";

interface FormData {
    username: string;
    password: string;
}

interface LoginProps {
```

```
    setIsLoggedIn: (value: boolean) => void;
}

const Login: React.FC<LoginProps> = ({ setIsLoggedIn }) => {
    useEffect(() => {
        window.scrollTo(0, 0);
    }, []);

    const [formData, setFormData] = useState<FormData>({
        username: "",
        password: "",
    });
    const [error, setError] = useState<string>("");
    const [authTime, setAuthTime] = useState<number | null>(null);
    const dispatch = useDispatch();
    const navigate = useNavigate();

    const handleChange = (e: ChangeEvent<HTMLInputElement>) => {
        const { name, value } = e.target;
        setFormData({
            ...formData,
            [name]: value,
        });
    };

    const handleSubmit = async (e: FormEvent<HTMLFormElement>) => {
        e.preventDefault();
        const startTime = Date.now(); // Початок вимірювання часу
        try {
            const response = await axios.post("http://localhost:8000/api/token/",
JSON.stringify(formData), {
                headers: {
                    "Content-Type": "application/json",
                },
            });
            const accessToken = response.data.access;
            localStorage.setItem("access_token", accessToken);
            dispatch(login());
```

```
            setIsLoggedIn(true);
            const endTime = Date.now(); // Кінець вимірювання часу
            setAuthTime(endTime - startTime); // Обчислення часу авторизації
            navigate(`/profile?authTime=${endTime - startTime}`); // Передача часу авторизації як параметра URL
          } catch (error) {
            setError("Неправильне ім'я користувача або пароль");
          }
        };

        return (
          <div className={styles.container}>
            <form className={styles.loginForm} onSubmit={handleSubmit}>
              <div className={styles.username}>
                <label htmlFor="username">Ім'я користувача:</label>
                <input className={styles.usernameInput} type="text" id="username" placeholder="Username" name="username" value={formData.username} onChange={handleChange} required />
              </div>
              <div className={styles.password}>
                <label htmlFor="password">Пароль:</label>
                <input className={styles.passwordInput} type="password" id="password" placeholder="Password" name="password" value={formData.password} onChange={handleChange} required />
              </div>
              <button className={styles.buttonLogin} type="submit">
                Логін
              </button>
              <Link to="/registration" className={styles.buttonRegistration}>
                Реєстрація
              </Link>
              {error && <p className={styles.error}>{error}</p>}
              {authTime !== null && <p className={styles.authTime}>Час авторизації: {authTime} мс</p>}
            </form>
          </div>
        );
      };
```

```
export default Login;
```

Login.module.css

```
/* Login.module.css */

.container {
    display: flex;
    align-items: center;
    justify-content: center;
    height: 100vh;
}

.loginForm {
    text-align: center;
    padding: 25px;
    margin: 10px;
    border: 5px solid #16ea1a;
    border-radius: 10px;
    font-family: Kanit, sans-serif;
    width: 300px;
}

.username,
.password {
    padding: 10px;
}

.usernameInput,
.passwordInput {
    background: #191b1f;
    border-radius: 5px;
    margin: 10px 0;
    border: 1px solid #ccc;
    color: #ffffff;
    font-size: 16px;
    padding: 8px;
    width: 100%;
    box-sizing: border-box;
```

```
}

.emailInput {
    margin: 10px 0;
}

.passwordInput {
    margin-top: 10px;
}

.usernameInput:focus,
.passwordInput:focus {
    outline: none;
    border-color: #16ea1a;
}

.buttonLogin,
.buttonRegistration {
    border: 1px solid #ffffff;
    border-radius: 5px;
    padding: 8px 15px;
    margin: 10px;
    background-color: #191b1f;
    color: #ffffff;
    font-size: 16px;
    transition: 0.35s;
    cursor: pointer;
}

.buttonLogin:hover {
    background-color: rgb(234, 222, 122);
    color: #191b1f;
    border: 1px solid #111111;
}

.buttonRegistration:hover {
    background-color: #16ea1a;
    color: #191b1f;
```

```
    border: 1px solid #111111;
}

.error {
    color: #ff0000;
    margin-top: 10px;
    font-size: 14px;
}

Pagination.tsx
import React from "react";
import styles from "./Pagination.module.css";

interface PaginationProps {
    totalPages: number;
    currentPage: number;
    onPageChange: (page: number) => void;
}

const Pagination: React.FC<PaginationProps> = ({ totalPages, currentPage, onPageChange }) =>
{
    const handlePrevClick = () => {
        if (currentPage > 1) {
            onPageChange(currentPage - 1);
        }
    };

    const handleNextClick = () => {
        if (currentPage < totalPages) {
            onPageChange(currentPage + 1);
        }
    };

    const renderPageNumbers = () => {
        const pageNumbers = [];
        for (let i = 1; i <= totalPages; i++) {
            pageNumbers.push(
                <li key={i}>
```

```
                    <button onClick={() => onPageChange(i)} className={i === currentPage ?
styles.currentPage : styles.pageNumber}>
                        {i}
                    </button>
                </li>
            );
        }
        return pageNumbers;
    };

    return (
        <div className={styles.paginationContainer}>
            <button onClick={handlePrevClick} disabled={currentPage === 1}
className={styles.paginationButton}>
                Попередня
            </button>
            <ul className={styles.pageNumbers}>{renderPageNumbers()}</ul>
            <button onClick={handleNextClick} disabled={currentPage === totalPages}
className={styles.paginationButton}>
                Далі
            </button>
        </div>
    );
};

export default Pagination;

Pagination.module.css
/* Pagination.module.css */

.paginationContainer {
    display: flex;
    justify-content: center;
    margin-top: 20px;
}

.paginationButton {
    background-color: #242424;
```

```
    color: #fff;
    border: none;
    border-radius: 10px;
    padding: 8px 16px;
    margin: 0 5px 20px;
    cursor: pointer;
    transition: background-color 0.3s;
}

.paginationButton:hover {
    background-color: #18b01a;
}

.pageNumbers {
    display: flex;
    list-style: none;
    padding: 0;
    margin: 0 10px;
}

.pageNumber {
    padding: 8px 16px;
    margin: 0 5px;
    border-radius: 10px;
    background-color: #2b2929;
    color: #fff;
    border-style: solid;
}

.currentPage {
    padding: 8px 16px;
    margin: 0 5px;
    border-radius: 10px;
    font-weight: bold;
    color: #fff;
    background-color: #00b354; /* Цвет активной кнопки */
    border-style: solid;
}
```

SingleProduct.tsx

```
import React, { useState, useEffect } from "react";
import { useParams } from "react-router-dom";
import BuyButton from "../buttons/BuyButton";
import axios from "axios";
import styles from "./SingleProduct.module.css";
import Slider from "react-slick"; //Імпортуємо бібліотеку для слайдера
import "slick-carousel/slick/slick.css";
import "slick-carousel/slick/slick-theme.css";

interface ProductData {
  id: number;
  name: string;
  price: number;
  description: string;
  image: string;
  count: number;
  attributes: { name: string; value: string }[];
  gallery: { image: string }[];
}

const SingleProduct: React.FC = () => {
  const { id = "" } = useParams<{ id: string }>();
  const [product, setProduct] = useState<ProductData | null>(null);
  const [loading, setLoading] = useState(true);
  const [activeImageIndex, setActiveImageIndex] = useState(0); // Стан для відстеження
поточного активного зображення

  useEffect(() => {
    const fetchData = async () => {
      try {
        const apiUrl = `http://127.0.0.1:8000/api/products/${id}/`;
        const response = await axios.get<ProductData>(apiUrl);
        setProduct(response.data);
        setLoading(false);
      } catch (error) {
        console.error("Помилка під час отримання даних:", error);
```

```
            setLoading(false);
        }
    };

    fetchData();
}, [id]);

if (loading) {
    return <div>Завантаження даних...</div>;
}

if (!product) {
    return <div>Товар не знайдено</div>;
}

// Функція для переключення активного зображення
const handleImageChange = (index: number) => {
    setActiveImageIndex(index);
};

const settings = {
    dots: true,
    infinite: true,
    speed: 500,
    slidesToShow: 1,
    slidesToScroll: 1,
};

return (
    <div className="container">
        <h1>{product.name}</h1>
        <div className="row">
            <div className="col">
                <div className={styles.slider}>
                    <Slider {...settings}>
                        {product.gallery.map((image, index) => (
                            <div key={index}>
```

```
                                <img className={styles.img} src={image.image} alt={product.name}
onClick={() => handleImageChange(index)} />
                                </div>
                            ))}
                        </Slider>
                    </div>

                    <div className={styles.attributes}>
                        <h2>Атрибути:</h2>
                        <ul>
                            {product.attributes.map((attribute, index) => (
                                <li key={index}>
                                    <strong>{attribute.name}:</strong> {attribute.value}
                                </li>
                            ))}
                        </ul>
                    </div>
                </div>

                <div className="col">
                    <h1 className={styles.price}>Ціна: {product.price} ₴</h1>
                    <p className={styles.description}>
                        <strong>Опис:</strong> {product.description}
                    </p>
                    <div className="row">
                        <div className="col">
                            <br />
                            <BuyButton product={product} />
                        </div>
                    </div>
                </div>
            </div>
        </div>
    );
};
export default SingleProduct;
SingleProduct.module.css
/* SingleProduct.module.css */
```

```
.img {
    width: 550px;
    height: 300px;
    justify-content: center;
    padding: 15px;
    border-radius: 25px;
}

.price {
    background-color: rgb(106, 202, 82);
    border-radius: 20px;
}

.description {
    font-size: 18px;
    margin: 10px;
}

.slider {
    max-width: 550px;
    max-height: 300px;
}
```

Product.tsx

```
// Product.tsx
import React, { useState, useEffect } from "react";
import { Link } from "react-router-dom";
import styles from "./Product.module.css";
import BuyButton from "../buttons/BuyButton";
import axios from "axios";
import Pagination from "../pagination/Pagination";

interface ProductProps {
    filter: string;
}

export interface ProductData {
    id: number;
```

```
    image: string;
    hovered_image: string;
    name: string;
    price: string;
    department: string;
    image_url: string;
    count: number;
}

const Product: React.FC<ProductProps> = ({ filter }) => {
    const [hoveredImg, setHoveredImg] = useState<number | null>(null);
    const [products, setProducts] = useState<ProductData[]>([]);
    const [loading, setLoading] = useState(true);
    const [currentPage, setCurrentPage] = useState(1);
    const [totalPages, setTotalPages] = useState(1);

    useEffect(() => {
        const fetchData = async () => {
            try {
                const apiUrl = `http://127.0.0.1:8000/api/products/?page=${currentPage}`;
                const response = await axios.get(apiUrl);
                setProducts(response.data.results);
                setTotalPages(Math.ceil(response.data.count / response.data.results.length));
                setLoading(false);
            } catch (error) {
                console.error("Помилка при отриманні даних:", error);
                setLoading(false);
            }
        };

        fetchData();
    }, [currentPage, filter]);

    const handlePageChange = (page: number) => {
        setCurrentPage(page);
    };

    const filterProducts = () => {
```

```
            switch (filter) {
              case "men":
                return products.filter((product) => product.department === "Men's");
              case "women":
                return products.filter((product) => product.department === "Women's");
              case "kids":
                return products.filter((product) => product.department === "Kid's");
              default:
                return products;
            }
          };

          return (
            <div>
              <div className={styles.productContainer}>
                {loading ? (
                  <p>Завантаження даних...</p>
                ) : Array.isArray(filterProducts()) ? (
                  filterProducts().map((product) => (
                    <form className={styles.productForm} key={product.id}>
                      <img className={styles.img} src={hoveredImg === product.id ?
product.hovered_image : product.image} alt={product.name} onMouseOver={() =>
setHoveredImg(product.id)} onMouseOut={() => setHoveredImg(null)} />
                      <Link key={`link-${product.id}`} to={`/singleProduct/${product.id}`}>
                        <h3 className={styles.name}>{product.name}</h3>
                      </Link>
                      <p>Ціна: {product.price} ₴</p>
                      <BuyButton product={product} />
                    </form>
                  ))
                ) : (
                  <p>Немає даних для відображення</p>
                )}
              </div>
              <div className={styles.paginationContainer}>
                <Pagination currentPage={currentPage} totalPages={totalPages}
onPageChange={handlePageChange} />
              </div>
```

```
        </div>
    );
};

export default Product;
```

Product.module.css

```
.img {
    width: 275px;
    height: 175px;
    justify-content: center;
    border-radius: 15px;
    padding: 5px;
}
.productContainer {
    width: 100%;
    display: flex;
    flex-wrap: wrap;
    justify-content: baseline;
}
.productForm {
    width: 25em;
    height: 32em;
    margin: 0.7%;
    border: 1px solid #100f0f;
    padding: 10px;
    text-align: center;
    border-radius: 10px;
    background-color: #100f0f;
    transition: background-color 0.3s;
}

.productForm:hover {
    background-color: #201d1dd4;
}

.size {
    font-size: 25px;
```

```
    font-family: Kanit, sans-serif;
}

.name {
    font-family: Kanit, sans-serif;
}

Profile.tsx
import React, { useState, useEffect } from "react";
import { useNavigate, useLocation } from "react-router-dom";
import Box from "@mui/material/Box";
import TextField from "@mui/material/TextField";
import axios from "axios";
import styles from "../profile/Profile.module.css";
import HeaderAuth from "../header/HeaderAuth"; // Імпорт HeaderAuth
import Header from "../header/Header";
import Button from "react-bootstrap/Button";
import Modal from "react-bootstrap/Modal";

const Profile: React.FC<{ isLoggedIn: boolean }> = ({ isLoggedIn }) => {
    const [userData, setUserData] = useState<any>(null);
    const [modalShow, setModalShow] = useState(false);
    const [authTime, setAuthTime] = useState<number | null>(null); // Додано стан для
збереження часу авторизації
    const navigate = useNavigate();
    const location = useLocation();

    // Отримання authTime з параметрів URL під час завантаження сторінки
    useEffect(() => {
        const searchParams = new URLSearchParams(location.search);
        const authTimeParam = searchParams.get("authTime");
        if (authTimeParam) {
            const authTime = parseInt(authTimeParam);
            setAuthTime(authTime);
        }
    }, [location.search]);

    useEffect(() => {
```

```
    const fetchUserData = async () => {
      try {
        const base64Token = localStorage.getItem("access_token");

        if (base64Token) {
          const tokenParts = base64Token.split(".");
          const payload = tokenParts[1];
          const decodedPayload = atob(payload);
          const parsedPayload = JSON.parse(decodedPayload);

          const userId = parsedPayload.user_id;

          const response = await axios.get(`http://localhost:8000/api/user/${userId}`, {
            headers: {
              Authorization: `JWT ${base64Token}`,
            },
          });

          setUserData(response.data);
        }
      } catch (error) {
        console.error("Помилка при отриманні даних користувача:", error);
      }
    };

    fetchUserData();
  }, []);

  const handleSaveChanges = async () => {
    try {
      const base64Token = localStorage.getItem("access_token");
      if (base64Token) {
        await axios.put(`http://localhost:8000/api/user/${userData.id}/`, userData, {
          headers: {
            Authorization: `JWT ${base64Token}`,
          },
        });
        setModalShow(true);
```

```
            }
        } catch (error) {
            console.error("Помилка при збереженні змін:", error);
        }
    };

    const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
        const { name, value } = e.target;
        setUserData((prevUserData: any) => ({
            ...prevUserData,
            [name]: value,
        }));
    };

    const handleLogout = () => {
        localStorage.removeItem("access_token");
        navigate("/login");
    };

    return (
        <div>
            {isLoggedIn ? <HeaderAuth isLoggedIn={isLoggedIn} /> : <Header
isLoggedIn={isLoggedIn} />}
            <div className={styles.navigate}></div>
            <h1>Профіль</h1>
            {/* Відображення authTime */}
            {authTime !== null && <p className={styles.authTime}>Час авторизації: {authTime}
мс</p>}
            {userData && (
                <Box
                    component="form"
                    sx={{
                        m: 1,
                        width: "25ch",
                        "& > :not(style)": {
                            margin: "7px 0",
                            "& .MuiInputBase-input": { color: "white" },
                            "& .MuiInputLabel-root": { color: "white !important" },
```

```
                        "& .MuiOutlinedInput-notchedOutline": {
                            borderColor: "white !important",
                        },
                        "&:hover .MuiOutlinedInput-notchedOutline": {
                            borderColor: "green !important",
                        },
                        "&:focus .MuiOutlinedInput-notchedOutline": {
                            borderColor: "green !important",
                        },
                        "& fieldset": { borderColor: "white !important" },
                    },
                }}
                noValidate
                autoComplete="off"
            >
                <TextField id="username" name="username" label="Ім'я користувача"
variant="outlined" defaultValue={userData.username} onChange={handleChange} />
                <TextField id="first_name" name="first_name" label="Ім'я" variant="outlined"
defaultValue={userData.first_name} onChange={handleChange} />
                <TextField id="last_name" name="last_name" label="Прізвище" variant="outlined"
defaultValue={userData.last_name} onChange={handleChange} />
            </Box>
        )}
        <div className={styles.buttonsContainer}>
            <button className={styles.customButton} onClick={handleSaveChanges}>
                <div className="svg-wrapper-1">
                    <div className="svg-wrapper">
                        <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24" width="30"
height="30" className="icon">
                            <path d="M22,15.04C22,17.23 20.24,19 18.07,19H5.93C3.76,19 2,17.23
2,15.04C2,13.07 3.43,11.44 5.31,11.14C5.28,11 5.27,10.86 5.27,10.71C5.27,9.33 6.38,8.2
7.76,8.2C8.37,8.2 8.94,8.43 9.37,8.8C10.14,7.05 11.13,5.44 13.91,5.44C17.28,5.44 18.87,8.06
18.87,10.83C18.87,10.94 18.87,11.06 18.86,11.17C20.65,11.54 22,13.13 22,15.04Z"></path>
                        </svg>
                    </div>
                </div>
                <span>Зберегти</span>
            </button>
```

```
                    <button className={styles.logoutButton} onClick={handleLogout}>
                        Вийти
                    </button>
                </div>

                {/* Модальне вікно */}
                <Modal show={modalShow} onHide={() => setModalShow(false)} size="lg" aria-
labelledby="contained-modal-title-vcenter" centered>
                    <Modal.Body className={styles.modal}>
                        <h4 className={styles.modal}>Зміни успішно збережено!</h4>
                    </Modal.Body>
                    <Modal.Footer className={styles.modal}>
                        <Button className={styles.modalButton} onClick={() => setModalShow(false)}>
                            Закрити
                        </Button>
                    </Modal.Footer>
                </Modal>

            </div>
        );
    };

    export default Profile;

    Profile.module.css
    /* Profile.module.css */

    .navigate {
        display: flex;
        justify-content: space-around;
        padding: 10px;
    }

    .navitem {
        text-decoration: none;
        color: #fff;
        padding: 10px;
        margin: 0 10px;
```

```
    border-bottom: 1px solid transparent;
}

.navitem:hover {
    text-decoration: none;
    border-bottom-color: #fff;
}

.customButton {
    margin: 10px;
    margin-left: 2.5em;
    font-family: inherit;
    font-size: 20px;
    background: #212121;
    color: white;
    fill: rgb(155, 153, 153);
    padding: 0.7em 1em;
    padding-left: 0.9em;
    display: flex;
    align-items: center;
    cursor: pointer;
    border: none;
    border-radius: 15px;
    font-weight: 1000;
}

.customButton span {
    display: block;
    margin-left: 0.3em;
    transition: all 0.3s ease-in-out;
}

.customButton svg {
    display: block;
    transform-origin: center center;
    transition: transform 0.3s ease-in-out;
}
```

```
.customButton:hover {
    background: #000;
}

.customButton:hover .svg-wrapper {
    transform: scale(1.25);
    transition: 0.5s linear;
}

.customButton:hover svg {
    transform: translateX(1.2em) scale(1.1);
    fill: #fff;
}

.customButton:hover span {
    opacity: 0;
    transition: 0.5s linear;
}

.customButton:active {
    transform: scale(0.95);
}

.logoutButton {
    margin: 10px;
    margin-left: 2.5em;
    font-family: inherit;
    font-size: 20px;
    background: #212121;
    color: white;
    fill: rgb(155, 153, 153);
    padding: 0.7em 1em;
    padding-left: 0.9em;
    display: flex;
    align-items: center;
    cursor: pointer;
    border: none;
    border-radius: 15px;
```

```
    font-weight: 1000;
}

.logoutButton:hover {
    background: #000;
}

.logoutButton:active {
    transform: scale(0.95);
}

.buttonsContainer {
    display: flex;
    justify-content: center;
    margin-top: 20px;
}

.modal {
    color: #fff;
    text-align: center;
    background-color: #292626;
}

.modalButton {
    background-color: rgb(106, 202, 82);
    margin: auto;
    display: block;
    border: 1px solid rgb(106, 202, 82);
}

.modalButton:hover {
    background-color: rgb(50, 125, 32);
    border: 1px solid rgb(50, 125, 32);
}
```

Search.tsx

```
// Search.tsx
import React, { useState, ChangeEvent, FormEvent, useEffect } from "react";
import { useDispatch, useSelector } from "react-redux";
import { useNavigate } from "react-router-dom";
import { setSearchTerm, setSearchResults, setSearchDuration } from "../../store/search.slice";
import { RootState } from "../../store/store";

import styles from "./Search.module.css";
import axios from "axios";

const Search: React.FC = () => {
  const [searchTermLocal, setSearchTermLocal] = useState<string>("");
  const navigate = useNavigate();
  const dispatch = useDispatch();
  const searchTerm = useSelector((state: RootState) => state.search.searchTerm);

  const handleInputChange = (e: ChangeEvent<HTMLInputElement>) => {
    setSearchTermLocal(e.target.value);
  };

  const handleSubmit = (e: FormEvent) => {
    e.preventDefault();
    dispatch(setSearchTerm(searchTermLocal));
    navigate("/SearchPage");
  };

  useEffect(() => {
    const fetchData = async () => {
      const startTime = Date.now();
      try {
        const apiUrl = `http://127.0.0.1:8000/api/products/?search=${searchTerm}`;
        const response = await axios.get(apiUrl);
        const endTime = Date.now();
        const duration = endTime - startTime;
        dispatch(setSearchResults(response.data.results));
        dispatch(setSearchDuration(duration));
      } catch (error) {
        console.error("Помилка під час пошуку:", error);
```

```
            }
        };

        if (searchTerm.trim() !== "") {
            fetchData();
        } else {
            dispatch(setSearchResults([]));
            dispatch(setSearchDuration(0));
        }
    }, [searchTerm, dispatch]);

    return (
        <form onSubmit={handleSubmit}>
            <input className={styles.search} type="text" placeholder="Пошук..."
value={searchTermLocal} onChange={handleInputChange} />
            <button className={styles.button} type="submit">
                Пошук
            </button>
        </form>
    );
};

export default Search;

Search.module.css
.search {
    background: #191b1f;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
    font-size: 16px;
    color: #ffffff;
    box-sizing: border-box;
}

.search:focus {
    outline: none;
    border-color: #16ea1a;
```

```
}

.button {
    font-family: Martian Mono, sans-serif;
    background-color: #000000;
    color: #fff;
    border: none;
    padding: 10px 20px;
    font-size: 16px;
    cursor: pointer;
    border-radius: 10px;
    text-decoration: none;
    margin: 5px;
}

.button:hover {
    background-color: #16ea1a;
    color: #000000;
}

@media (max-width: 768px) {
    .search {
        font-size: 16px;
        padding: 5px;
    }
}
```

SearchPage.tsx

```
// SearchPage.tsx
import { Link } from "react-router-dom";
import { useSelector } from "react-redux";
import { RootState } from "../../store/store";
import BuyButton from "../buttons/BuyButton";
import styles from "./Search.module.css";

const SearchPage = () => {
    const searchResults = useSelector((state: RootState) => state.search.searchResults);
    const searchDuration = useSelector((state: RootState) => state.search.searchDuration);
```

```
    return (
      <div className={styles.search}>
        <h2>Результати пошуку:</h2>
        <p>Час пошуку: {searchDuration.toFixed(3)} мс</p>
        <div className={styles.productContainer}>
          {Array.isArray(searchResults) && searchResults.length > 0 ? (
            searchResults.map((result) => (
              <form className={styles.productForm} key={result.id}>
                <img className={styles.img} src={result.image} alt={result.name} />
                <Link key={`link-${result.id}`} to={`/singleProduct/${result.id}`}>
                  <h3 className={styles.name}>{result.name}</h3>
                </Link>
                <p>Ціна: {result.price}</p>

                <br />
                <BuyButton product={result} />
              </form>
            ))
          ) : (
            <div className={styles.notFoundContainer}>
              <div className="row">
                <h1 className={`${styles.nothing_found} col`}>Нічого не знайдено</h1>
                <img src="search-not-found-1.png" alt="Нічого не знайдено"
```

```
className="col-5" />
              </div>
            </div>
          )}
        </div>
      </div>
    );
  };

  export default SearchPage;

  Search.module.css
  .img {
    width: 275px;
```

```
    height: 175px;
    justify-content: center;
    border-radius: 15px;
    padding: 5px;
}

.productContainer {
    display: flex;
    flex-wrap: wrap;
    justify-content: baseline;
}
.productForm {
    width: 450px;
    height: 470px;
    margin: 10px;
    border: 1px solid #100f0f;
    padding: 10px;
    text-align: center;
    border-radius: 10px;
    background-color: #100f0f;
    transition: background-color 0.3s;
}

.productForm:hover {
    background-color: #201d1dd4;
}

.size {
    font-size: 25px;
    font-family: Kanit, sans-serif;
}

.name {
    font-family: Kanit, sans-serif;
}

.nothing_found {
    display: flex;
```

```
    align-items: center;
    justify-content: center;
    height: 100vh;
    text-align: center;
}

.notFoundImage_1 {
    height: 500px;
    width: 500px;
}

.search {
    overflow: hidden;
}
```

ShoeCategories.tsx

```
import { NavLink } from "react-router-dom";
import styles from "./ShoeCategories.module.css";
import { motion } from "framer-motion";

const ShoeCategories: React.FC = () => {
    return (
        <>
            <motion.h1
                initial={{
                    y: -1000,
                    opacity: 0,
                }}
                animate={{ y: 0, opacity: 1 }}
                transition={{ delay: 0.1 }}
            >
                Виберіть категорію
            </motion.h1>
            <nav className={styles.tileContainer}>
                <motion.div
                    initial={{
                        x: -1000,
                        opacity: 0,
```

```
            }}
            animate={{ x: 0, opacity: 1 }}
            transition={{ delay: 0.5 }}
          >
            <NavLink to={"/men"} className={`${styles.tile} ${styles.man}`}>
              <h3 className={styles.container}>Чоловіки</h3>
            </NavLink>
          </motion.div>
          <motion.div
            initial={{
              x: -1000,
              opacity: 0,
            }}
            animate={{ x: 0, opacity: 1 }}
            transition={{ delay: 0.2 }}
          >
            <NavLink to={"/women"} className={`${styles.tile} ${styles.woman}`}>
              <h3 className={styles.container}>Жінки</h3>
            </NavLink>
          </motion.div>

          <motion.div
            initial={{
              x: -1000,
              opacity: 0,
            }}
            animate={{ x: 0, opacity: 1 }}
            transition={{ delay: 0.3 }}
          >
            <NavLink to={"/kids"} className={`${styles.tile} ${styles.child}`}>
              <h3 className={styles.container}>Діти</h3>
            </NavLink>
          </motion.div>
        </nav>
      </>
    );
};
```

```
export default ShoeCategories;

ShoeCategories.module.css
/* ShoeCategories.module.css */

.tileContainer {
    display: flex;
    justify-content: center;
    flex-wrap: wrap;
    padding: 1em;
}

.tile {
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    border: 1px solid #ccc;
    padding: 16px;
    margin: 8px;
    text-align: center;
    text-decoration: none;
    color: #ffffff;
    width: 22em;
    height: 22em;
}

.man {
    background-image: url(https://marvel-b1-
cdn.bc0a.com/f00000000114841/www.florsheim.com/styleguide/resources/images/about/styleTips/FW22
-FL-BlogUpdate-TheseAreTheSixShoesMenNeed-ArticleImage2-Desktop.jpg);
    background-size: cover;
}

.woman {
    background-image: url(https://images.pexels.com/photos/2285500/pexels-photo-
2285500.jpeg?auto=compress&cs=tinysrgb&dpr=1&w=500);
    background-size: cover;
```

```
}

.child {
    background-image: url(https://media.istockphoto.com/id/518868426/photo/new-sneakers-on-boys-feet.jpg?s=612x612&w=0&k=20&c=GSlpksRijdwFfVRvIL8LZkkwmlKUgLPduQ8Y8HTzQIo=);
    background-size: cover;
}

.tile .container {
    background-color: rgb(0, 0, 0);
    border-radius: 5px;
    padding: 20px;
    transition: transform 0.3s, box-shadow 0.3s, background-color 0.3s;
}

.tile .container:hover {
    color: #333;
    transform: scale(1.1);
    box-shadow: 0 0 20px rgba(0, 0, 0, 0.5);
    background-color: rgb(104, 220, 141);
    -webkit-transform: rotateZ(10deg);
    -ms-transform: rotateZ(10deg);
    transform: rotateZ(10deg);
}

NotFound.tsx
const NotFound = () => {
    return (
        <div style={{ maxWidth: "100%", maxHeight: "100%", overflow: "hidden" }}>
            <img src="404.jpg" alt="Not Found" style={{ maxWidth: "100%", maxHeight: "100%" }} />
        </div>
    );
};

export default NotFound;
```

auth.slice.ts

```
// authSlice.ts
import { createSlice } from "@reduxjs/toolkit";

interface AuthState {
   isLoggedIn: boolean;
}

const initialState: AuthState = {
   isLoggedIn: false,
};

const authSlice = createSlice({
   name: "auth",
   initialState,
   reducers: {
      login(state) {
         state.isLoggedIn = true;
      },
      logout(state) {
         state.isLoggedIn = false;
      },
   },
});

export const { login, logout } = authSlice.actions;
export default authSlice.reducer;
```

cart.slice.ts

```
import { createSlice, PayloadAction } from "@reduxjs/toolkit";

export interface CartItem {
   id: number;
   name: string;
   price: string;
   image_url: string;
```

```
    count: number;
    image: string;
}

export interface CartState {
    items: CartItem[];
}

const initialState: CartState = {
    items: [],
};

export const cartSlice = createSlice({
    name: "cart",
    initialState,
    reducers: {
        add: (state, action: PayloadAction<CartItem>) => {
            const { id } = action.payload;
            const existingItemIndex = state.items.findIndex((item) => item.id === id);
            if (existingItemIndex !== -1) {
                state.items[existingItemIndex].count += 1;
            } else {
                state.items.push({ ...action.payload, count: 1 });
            }
        },
        remove: (state, action: PayloadAction<number>) => {
            state.items = state.items.filter((item) => item.id !== action.payload);
        },
        decrease: (state, action: PayloadAction<number>) => {
            const itemToDecrease = state.items.find((item) => item.id === action.payload);
            if (itemToDecrease && itemToDecrease.count > 1) {
                itemToDecrease.count -= 1;
            }
        },
        incrementQuantity: (state, action: PayloadAction<number>) => {
            const itemToIncrement = state.items.find((item) => item.id === action.payload);
            if (itemToIncrement) {
                itemToIncrement.count += 1;
```

```
        }
      },
    },
});

export const cartActions = cartSlice.actions;
export default cartSlice.reducer;
```

search.slice.ts

```
// searchSlice.ts
import { createSlice, PayloadAction } from "@reduxjs/toolkit";

interface Product {
  id: string;
  image: string;
  name: string;
  price: number;
}

interface SearchState {
  searchTerm: string;
  searchResults: Product[];
  searchDuration: number;
}

const initialState: SearchState = {
  searchTerm: "",
  searchResults: [],
  searchDuration: 0,
};

const searchSlice = createSlice({
  name: "search",
  initialState,
  reducers: {
    setSearchTerm(state, action: PayloadAction<string>) {
      state.searchTerm = action.payload;
    },
```

```
        setSearchResults(state, action: PayloadAction<Product[]>) {
          state.searchResults = action.payload;
        },
        setSearchDuration(state, action: PayloadAction<number>) {
          state.searchDuration = action.payload;
        },

        updateSearchDuration(state, action: PayloadAction<number>) {
          state.searchDuration = action.payload;
        },
      },
    });

    export const { setSearchTerm, setSearchResults, setSearchDuration, updateSearchDuration } =
searchSlice.actions;
    export default searchSlice.reducer;

    store.ts
    import { configureStore } from "@reduxjs/toolkit";
    import cartSlice from "./cart.slice";
    import authSlice from "./auth.slice";
    import searchReducer from "./search.slice";

    export const store = configureStore({
      reducer: {
        cart: cartSlice,
        auth: authSlice,
        search: searchReducer,
      },
    });

    export type RootState = ReturnType<typeof store.getState>;
    export type AppDispath = typeof store.dispatch;
    manage.py
    #!/usr/bin/env python
    """Django's command-line utility for administrative tasks."""
    import os
    import sys
```

```
def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'base.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

asgi.py

```
"""
ASGI config for base project.

It exposes the ASGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/howto/deployment/asgi/
"""

import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'base.settings')

application = get_asgi_application()
```

settings.py

```
"""
Django settings for base project.

Generated by 'django-admin startproject' using Django 4.2.6.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/ref/settings/
"""

from pathlib import Path
from datetime import timedelta

import yaml

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = "django-insecure-)!+k$_5s4m!44@q6((53r+7f(pns50r=-av!*yj54@%*4(lztp"

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
```

```
    "django.contrib.messages",
    "django.contrib.staticfiles",
    # In project apps
    "restapi.apps.RestapiConfig",
    # REST & CORS
    "rest_framework",
    'rest_framework_simplejwt',
    'drf_spectacular',
    "corsheaders",
]

MIDDLEWARE = [
    "django.middleware.security.SecurityMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
    # CORS
    "corsheaders.middleware.CorsMiddleware",
    "django.middleware.common.CommonMiddleware",
]

# CORS
CORS_ALLOWED_ORIGINS = [
    "http://localhost:5173",
    "http://localhost:8000",
    "http://127.0.0.1:5173",
    "http://127.0.0.1:8000",
]

ROOT_URLCONF = "base.urls"

TEMPLATES = [
    {
        "BACKEND": "django.template.backends.django.DjangoTemplates",
        "DIRS": [],
```

```
        "APP_DIRS": True,
        "OPTIONS": {
            "context_processors": [
                "django.template.context_processors.debug",
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",
                "django.contrib.messages.context_processors.messages",
            ],
        },
    },
]

WSGI_APPLICATION = "base.wsgi.application"

# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

# DATABASES = {
#     'default': {
#         'ENGINE': 'django.db.backends.sqlite3',
#         'NAME': BASE_DIR / 'db.sqlite3',
#     }
# }

database_config = yaml.load(open("database.config.yaml"), Loader=yaml.Loader)

DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.mysql",
        "NAME": database_config["NAME"],
        "USER": database_config["USER"],
        "PASSWORD": database_config["PASSWORD"],
        "HOST": database_config["HOST"],
        "PORT": database_config["PORT"],
    }
}

# Password validation
```

```
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        "NAME": "django.contrib.auth.password_validation.UserAttributeSimilarityValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.MinimumLengthValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.CommonPasswordValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.NumericPasswordValidator",
    },
]

# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE_CODE = "en-us"

TIME_ZONE = "UTC"

USE_I18N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = "static/"

MEDIA_URL = '/media/'

MEDIA_ROOT = BASE_DIR / 'uploads'

# Default primary key field type
```

```
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"

# Set up pagination
REST_FRAMEWORK = {
    'DEFAULT_SCHEMA_CLASS': 'drf_spectacular.openapi.AutoSchema',
    'DEFAULT_PERMISSION_CLASSES': ['rest_framework.permissions.AllowAny'],
    'DEFAULT_PAGINATION_CLASS': "rest_framework.pagination.PageNumberPagination",
    'PAGE_SIZE': 6,
    'DEFAULT_FILTER_BACKENDS': [
        'django_filters.rest_framework.DjangoFilterBackend',
    ],
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    )
}

SPECTACULAR_SETTINGS = {
    'TITLE': 'StyleStepHaven',
    'DESCRIPTION': '"
    Peculiarities
    Variety of styles: The store offers a wide selection of shoes that meet the latest fashion trends. Whether you're looking for casual sneakers, stylish heels or comfortable boots, we have something for everyone.
    Filtering and search: Convenient filtering and search tools will help you quickly find exactly the models that interest you.
    Detailed Product Descriptions: Each shoe comes with detailed descriptions, sizes, colors and photos so you can make an informed decision.
    ",
    'VERSION': "1.0.0-alpha",
    'SERVE_INCLUDE_SCHEME': False,
}

SIMPLE_JWT = {
    "ACCESS_TOKEN_LIFETIME": timedelta(minutes=25),
    "REFRESH_TOKEN_LIFETIME": timedelta(days=1),
    "ROTATE_REFRESH_TOKENS": False,
```

```
    "BLACKLIST_AFTER_ROTATION": False,
    "UPDATE_LAST_LOGIN": False,

    "ALGORITHM": "HS256",
    "SIGNING_KEY": SECRET_KEY,
    "VERIFYING_KEY": "",
    "AUDIENCE": None,
    "ISSUER": None,
    "JSON_ENCODER": None,
    "JWK_URL": None,
    "LEEWAY": 0,

    "AUTH_HEADER_TYPES": ("JWT",),  # "AUTH_HEADER_TYPES": ("Bearer",), is default
    "AUTH_HEADER_NAME": "HTTP_AUTHORIZATION",
    "USER_ID_FIELD": "id",
    "USER_ID_CLAIM": "user_id",
    "USER_AUTHENTICATION_RULE":
"rest_framework_simplejwt.authentication.default_user_authentication_rule",

    "AUTH_TOKEN_CLASSES": ("rest_framework_simplejwt.tokens.AccessToken",),
    "TOKEN_TYPE_CLAIM": "token_type",
    "TOKEN_USER_CLASS": "rest_framework_simplejwt.models.TokenUser",

    "JTI_CLAIM": "jti",

    "SLIDING_TOKEN_REFRESH_EXP_CLAIM": "refresh_exp",
    "SLIDING_TOKEN_LIFETIME": timedelta(minutes=25),
    "SLIDING_TOKEN_REFRESH_LIFETIME": timedelta(days=1),

    "TOKEN_OBTAIN_SERIALIZER":
"rest_framework_simplejwt.serializers.TokenObtainPairSerializer",
    "TOKEN_REFRESH_SERIALIZER":
"rest_framework_simplejwt.serializers.TokenRefreshSerializer",
    "TOKEN_VERIFY_SERIALIZER":
"rest_framework_simplejwt.serializers.TokenVerifySerializer",
    "TOKEN_BLACKLIST_SERIALIZER":
"rest_framework_simplejwt.serializers.TokenBlacklistSerializer",
```

```
    "SLIDING_TOKEN_OBTAIN_SERIALIZER":
"rest_framework_simplejwt.serializers.TokenObtainSlidingSerializer",
    "SLIDING_TOKEN_REFRESH_SERIALIZER":
"rest_framework_simplejwt.serializers.TokenRefreshSlidingSerializer",
}

urls.py
"""
URL configuration for base project.

The `urlpatterns` list routes URLs to views. For more information please see:
  https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
  1. Add an import:  from my_app import views
  2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
  1. Add an import:  from other_app.views import Home
  2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
  1. Import the include() function: from django.urls import include, path
  2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.conf import settings
from django.conf.urls.static import static
from django.contrib import admin
from django.urls import path, include
from drf_spectacular.views import (
  SpectacularSwaggerView,
  SpectacularRedocView
)
from rest_framework_simplejwt.views import (
  TokenObtainPairView,
  TokenRefreshView,
  TokenVerifyView,
)
```

```
from .views import index, SchemaAPIView

urlpatterns = [
    path('', index, name="index"),
    path('admin/', admin.site.urls),
    path('api/schema/', SchemaAPIView.as_view(), name='schema'),
    path('api/schema/redoc/', SpectacularRedocView.as_view(url_name="schema"),
name='redoc'),
    path('api/schema/swagger/', SpectacularSwaggerView.as_view(url_name="schema"),
name='swagger'),
    path('api/', include("restapi.urls"), name="api"),
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
    path('api/token/verify/', TokenVerifyView.as_view(), name='token_verify'),
]

if settings.DEBUG:

    urlpatterns.extend(
        static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
    )

    urlpatterns.extend(
        static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
    )

views.py
from django.http import HttpRequest, HttpResponse
from django.shortcuts import redirect
from django.urls import reverse
from drf_spectacular.views import SpectacularAPIView
from drf_spectacular.utils import extend_schema


def index(request: HttpRequest) -> HttpResponse:
    return redirect("api/")
```

```
@extend_schema(
    summary="Retrieve Schema"
)
class SchemaAPIView(SpectacularAPIView):
    pass
```

wsgi.py

```
"""
WSGI config for base project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'base.settings')

application = get_wsgi_application()
```

admin.py

```
from django.contrib import admin

from .models import Product, ProductImage, ProductAttribute, ProductSize


class ProductImageInline(admin.StackedInline):
    model = ProductImage


class ProductAttributeInline(admin.StackedInline):
    model = ProductAttribute
```

```
class ProductSizeInline(admin.StackedInline):
    model = ProductSize


@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    inlines = (ProductAttributeInline, ProductImageInline, ProductSizeInline)
```

apps.py

```
from django.apps import AppConfig


class RestapiConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'restapi'
```

models.py

```
from enum import Enum

from django.contrib.auth.models import User
from django.db import models


def get_product_images_upload_location(instance: "Product", filename: str) -> str:
    return f"products/preview/images/{filename}"


class Department(Enum):
    MEN = "Men's"
    WOMEN = "Women's"
    KID = "Kid's"


# Create your models here.
class Product(models.Model):
    name = models.CharField(max_length=128)
    description = models.TextField()
```

```
        department = models.CharField(max_length=32, choices=[(tag.value, tag.name) for tag in
Department])
        price = models.DecimalField(default=0, max_digits=8, decimal_places=2)
        image = models.ImageField(null=True, blank=True,
upload_to=get_product_images_upload_location)
        hovered_image = models.ImageField(null=True, blank=True,
upload_to=get_product_images_upload_location)
        created_at = models.DateTimeField(auto_now_add=True)

        def __str__(self):
            return str(self.name)


    def get_product_gallery_upload_location(instance: "ProductImage", filename: str) -> str:
        return f"products/gallery/{filename}"


    class ProductImage(models.Model):
        product = models.ForeignKey(
            to=Product,
            on_delete=models.CASCADE,
            related_name="images",  # source='images' in Serializer
            verbose_name="images",
            null=True,
        )
        image = models.ImageField(
            upload_to=get_product_gallery_upload_location
        )


    class ProductAttribute(models.Model):
        product = models.ForeignKey(
            to=Product,
            on_delete=models.CASCADE,
            related_name="attributes",
            verbose_name="attributes",
            null=True,
        )
```

```
    name = models.CharField(max_length=128, null=False)
    value = models.CharField(max_length=128, null=False)


class Status(Enum):
    AWAITING_CONFIRMATION = 'AWAITING CONFIRMATION'
    PAYED = 'PAYED'
    CONFIRMED = 'CONFIRMED'
    IN_PROCESSING = 'IN PROCESSING'
    SEND = 'SEND'
    DELIVERED = 'DELIVERED'
    REJECTED = 'REJECTED'
    CANCELLED = 'CANCELLED'
    RETURN = 'RETURN'

class ProductSize(models.Model):
    product = models.ForeignKey(
        to=Product,
        on_delete=models.CASCADE,
        related_name="sizes",
        verbose_name="sizes",
        null=True,
    )
    size = models.IntegerField()


class Profile(models.Model):
    user = models.OneToOneField(to=User, on_delete=models.CASCADE)
    city = models.CharField(max_length=128, null=True, blank=True)
    phone_number = models.CharField(max_length=12, null=True, blank=True)
```

purchase.py

```
from cloudipsp import Api, Checkout


class Purchase:
    def __init__(self, integer: int, decimal: str):
        self._integer: str = str(integer)
```

```
        self._decimal: str = str(decimal)

    def get_url(self):
        """source: https://github.com/cloudipsp/python-sdk"""

        api: Api = Api(merchant_id=1396424, secret_key="test")
        checkout: Checkout = Checkout(api=api)
        data: dict = {
            "currency": "UAH",
            "amount": self._integer + self._decimal,
        }
        url: str = checkout.url(data).get("checkout_url")
        return url
```

serializer.py

```
from django.contrib.auth.hashers import make_password
from django.contrib.auth.models import User
from rest_framework import serializers

from .models import (
    Product,
    ProductImage,
    ProductAttribute,
    ProductSize,
    Profile
)


class ProductImageSerializer(serializers.ModelSerializer):
    class Meta:
        model = ProductImage
        fields = ['image']


class ProductAttributeSerializer(serializers.ModelSerializer):
    class Meta:
        model = ProductAttribute
        fields = ['name', 'value']
```

```
class ProductSizeSerializer(serializers.ModelSerializer):
    class Meta:
        model = ProductSize
        fields = ['size']


class ProductSerializer(serializers.ModelSerializer):
    gallery = ProductImageSerializer(many=True, read_only=True, source="images")
    attributes = ProductAttributeSerializer(many=True, read_only=True)  # source="attributes" is redundant!
    sizes = ProductSizeSerializer(many=True, read_only=True)

    class Meta:
        model = Product
        fields = (
            "id",
            "name",
            "description",
            "department",
            "price",
            "image",
            "hovered_image",
            "sizes",
            "attributes",
            "gallery"
        )


class UserProfileSerializer(serializers.ModelSerializer):

    class Meta:
        model = Profile
        fields = (
            "city",
            "phone_number",
        )
```

```
class UserSerializer(serializers.ModelSerializer):

    profile = UserProfileSerializer(many=False, read_only=True)

    class Meta:
        model = User
        fields = (
            "id",
            "username",
            "first_name",
            "last_name",
            "date_joined",
            "last_login",
            "profile",
        )


class UserRegisterSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = (
            "id",
            "username",
            "first_name",
            "last_name",
            "password",
        )
        read_only_fields = ("id",)  # Mark 'id' as read-only to prevent modification

    def create(self, validated_data):
        # Hash password before save.
        validated_data['password'] = make_password(validated_data['password'])
        user = super(UserRegisterSerializer, self).create(validated_data)
        return user
```

urls.py

```
from django.urls import path, include
from rest_framework.routers import DefaultRouter

from .views import ProductViewSet, PurchaseView, UserView, UserCreateView

app_name = "restapi"

router = DefaultRouter()
router.register(prefix="products", viewset=ProductViewSet)
router.register(prefix="products", viewset=ProductViewSet)

urlpatterns = [
    path('', include(router.urls), name='api_root'),
    path('user/create/', UserCreateView.as_view({'post': 'create'}), name="user_create"),
    path('user/<int:pk>/', UserView.as_view({'get': 'retrieve', 'put': 'update'}), name="user"),
    path('submit-order/', PurchaseView.as_view(), name="submit_order"),
]
```

views.py

```
from typing import Any, Optional
from collections import OrderedDict

from django.contrib.auth.models import User
from rest_framework import status
from rest_framework.filters import SearchFilter
from rest_framework.permissions import IsAuthenticated
from rest_framework.views import APIView
from rest_framework.viewsets import ModelViewSet, GenericViewSet
from rest_framework.mixins import RetrieveModelMixin, CreateModelMixin,
UpdateModelMixin
from rest_framework.request import Request
from rest_framework.response import Response
from drf_spectacular.utils import extend_schema, OpenApiResponse

from .models import Product, ProductImage, Profile
from .serializer import (
    ProductSerializer,
```

```
    ProductImageSerializer,
    UserSerializer,
    UserRegisterSerializer,
    UserProfileSerializer,
)
from .purchase import Purchase


class ProductViewSet(ModelViewSet):
    queryset = Product.objects.all()
    serializer_class = ProductSerializer
    filter_backends = [
        SearchFilter,
    ]
    search_fields = ["name", "description"]

    @extend_schema(
        summary="Create product"
    )
    def create(self, *args: Any, kwargs: Any) -> Response:
        return super().create(*args, kwargs)

    @extend_schema(
        summary="Get all products"
    )
    def list(self, *args: Any, kwargs: Any) -> Response:
        return super().list(*args, kwargs)

    @extend_schema(
        summary="Update products"
    )
    def update(self, *args: Any, kwargs: Any) -> Response:
        return super().update(*args, kwargs)

    @extend_schema(
        summary="Partial update products"
    )
    def partial_update(self, *args: Any, kwargs: Any) -> Response:
```

```
        return super().partial_update(*args, kwargs)

    @extend_schema(
        summary="Destroy product"
    )
    def destroy(self, *args: Any, kwargs: Any) -> Response:
        return super().destroy(*args, kwargs)

    @extend_schema(
        summary="Get one product by ID",
        description="Retrieves one product, returns 404 if not found!",
        responses={
            200: ProductSerializer,
            404: OpenApiResponse(
                description="Empty response, product by id not found."
            )
        }
    )
    def retrieve(self, *args: Any, kwargs: Any) -> Response:
        return super().retrieve(*args, kwargs)



@extend_schema(
    summary="Submit order"
)
class PurchaseView(APIView):

    def post(self, request: Request) -> Response:
        amount: float = 0.0
        products = request.data.get("order")
        for product in products:
            amount += float(product.get("price")) * int(product.get("count"))

        amount_str: str = str(amount)

        integer = int(amount)
        decimal = amount_str[-2:].replace(".", "")
```

```
        if len(decimal) == 1:
            decimal += "0"

        print(integer, decimal)

        purchase = Purchase(integer=integer, decimal=decimal)
        url = purchase.get_url()

        return Response({"url": url})


class UserCreateView(GenericViewSet, CreateModelMixin):
    serializer_class = UserRegisterSerializer

    def create(self, request, *args, kwargs):
        # Create user
        response = super().create(request, *args, kwargs)

        # If user created successfully
        if response.status_code == status.HTTP_201_CREATED:
            user_data = response.data

            # Get or create the user
            user, created = User.objects.get_or_create(
                id=user_data['id'],
                username=user_data['username'],
                first_name=user_data['first_name'],
                last_name=user_data['last_name'],
                password=user_data['password']
            )

            # Create profile for the user
            Profile.objects.create(user=user)

        return response
```

```
        class UserView(GenericViewSet, CreateModelMixin, RetrieveModelMixin,
UpdateModelMixin):
            queryset = User.objects.all()
            serializer_class = UserSerializer
            permission_classes = (
                IsAuthenticated,
            )

            def get_object(self) -> Optional[User]:
                # Get object from URL
                user = super().get_object()

                # Get authenticated user
                authenticated_user = self.request.user

                if user.id != authenticated_user.id:
                    return None

                return user

            def update(self, request, *args, kwargs):
                response = super().update(request, *args, kwargs)

                # If user updated successfully
                if response.status_code == status.HTTP_200_OK:
                    user_data = response.data

                    # Get profile data from request
                    profile_data = request.data.get('profile', {})
                    user_profile = self.get_object().profile

                    # Update profile if profile data is provided
                    if profile_data:
                        profile_serializer = UserProfileSerializer(instance=user_profile, data=profile_data,
partial=True)
                        if profile_serializer.is_valid():
                            profile_serializer.save()
                        else:
```

```
                return Response(profile_serializer.errors,
status=status.HTTP_400_BAD_REQUEST)

        return response
```

0001_initial.py

```
# Generated by Django 4.2.7 on 2023-11-14 16:20

from django.db import migrations, models


class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='CatBreed',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
                ('breed_name', models.CharField(max_length=100)),
                ('breed_description', models.TextField()),
                ('url_to_image', models.TextField()),
            ],
        ),
    ]
```

0002_product_delete_catbreed.py

```
# Generated by Django 4.2.7 on 2023-11-24 08:03

from django.db import migrations, models


class Migration(migrations.Migration):
```

```
        dependencies = [
            ('restapi', '0001_initial'),
        ]

        operations = [
            migrations.CreateModel(
                name='Product',
                fields=[
                    ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
                    ('name', models.CharField(max_length=128)),
                    ('description', models.TextField()),
                    ('price', models.DecimalField(decimal_places=2, default=0, max_digits=8)),
                    ('image_url', models.CharField(max_length=128)),
                    ('hovered_image_url', models.CharField(max_length=128)),
                ],
            ),
            migrations.DeleteModel(
                name='CatBreed',
            ),
        ]
```

0003_product_created_at_product_department.py

```
# Generated by Django 4.2.7 on 2023-11-24 08:27

from django.db import migrations, models
import django.utils.timezone


class Migration(migrations.Migration):

    dependencies = [
        ('restapi', '0002_product_delete_catbreed'),
    ]

    operations = [
        migrations.AddField(
```

```
            model_name='product',
            name='created_at',
            field=models.DateTimeField(auto_now_add=True, default=django.utils.timezone.now),
            preserve_default=False,
        ),
        migrations.AddField(
            model_name='product',
            name='department',
            field=models.CharField(max_length=128, null=True),
        ),
    ]
```

0004_remove_product_hovered_image_url_and_more.py

```
# Generated by Django 4.2.7 on 2024-01-12 13:14

from django.db import migrations, models
import restapi.models


class Migration(migrations.Migration):

    dependencies = [
        ('restapi', '0003_product_created_at_product_department'),
    ]

    operations = [
        migrations.RemoveField(
            model_name='product',
            name='hovered_image_url',
        ),
        migrations.RemoveField(
            model_name='product',
            name='image_url',
        ),
        migrations.AddField(
            model_name='product',
            name='hovered_image',
```

```
                field=models.ImageField(blank=True, null=True,
upload_to=restapi.models.get_product_images_upload_location),
            ),
            migrations.AddField(
                model_name='product',
                name='image',
                field=models.ImageField(blank=True, null=True,
upload_to=restapi.models.get_product_images_upload_location),
            ),
        ]
```

0005_alter_product_hovered_image_alter_product_image_and_more.py

```
# Generated by Django 4.2.7 on 2024-01-21 13:36

from django.db import migrations, models
import django.db.models.deletion
import restapi.models


class Migration(migrations.Migration):

    dependencies = [
        ('restapi', '0004_remove_product_hovered_image_url_and_more'),
    ]

    operations = [
        migrations.AlterField(
            model_name='product',
            name='hovered_image',
            field=models.ImageField(blank=True, null=True,
upload_to=restapi.models.get_product_images_upload_location),
        ),
        migrations.AlterField(
            model_name='product',
            name='image',
            field=models.ImageField(blank=True, null=True,
upload_to=restapi.models.get_product_images_upload_location),
        ),
```

```
            migrations.CreateModel(
                name='ProductImage',
                fields=[
                    ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
                    ('image',
models.ImageField(upload_to=restapi.models.get_product_images_upload_location)),
                    ('product', models.ForeignKey(null=True,
on_delete=django.db.models.deletion.CASCADE, related_name='images', to='restapi.product',
verbose_name='images')),
                ],
            ),
        ]
```

0006_alter_product_hovered_image_alter_product_image.py

```
# Generated by Django 4.2.7 on 2024-01-21 13:36

from django.db import migrations, models
import restapi.models


class Migration(migrations.Migration):

    dependencies = [
        ('restapi', '0005_alter_product_hovered_image_alter_product_image_and_more'),
    ]

    operations = [
        migrations.AlterField(
            model_name='product',
            name='hovered_image',
            field=models.ImageField(blank=True, null=True,
upload_to=restapi.models.get_product_images_upload_location),
        ),
        migrations.AlterField(
            model_name='product',
            name='image',
```

```
                field=models.ImageField(blank=True, null=True,
upload_to=restapi.models.get_product_images_upload_location),
            ),
        ]
```

0007_productattribute.py

```
# Generated by Django 4.2.7 on 2024-01-21 14:06

from django.db import migrations, models
import django.db.models.deletion


class Migration(migrations.Migration):

    dependencies = [
        ('restapi', '0006_alter_product_hovered_image_alter_product_image'),
    ]

    operations = [
        migrations.CreateModel(
            name='ProductAttribute',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
                ('name', models.CharField(max_length=128)),
                ('value', models.CharField(max_length=128)),
                ('product', models.ForeignKey(null=True,
on_delete=django.db.models.deletion.CASCADE, related_name='attributes', to='restapi.product',
verbose_name='attributes')),
            ],
        ),
    ]
```

0008_alter_product_department_alter_productimage_image_and_more.py

```
# Generated by Django 4.2.7 on 2024-01-24 10:43

from django.conf import settings
from django.db import migrations, models
```

```
import django.db.models.deletion
import restapi.models


class Migration(migrations.Migration):

    dependencies = [
        migrations.swappable_dependency(settings.AUTH_USER_MODEL),
        ('restapi', '0007_productattribute'),
    ]

    operations = [
        migrations.AlterField(
            model_name='product',
            name='department',
            field=models.CharField(choices=[("Men's", 'MEN'), ("Women's", 'WOMEN'), ("Kid's",
'KID')], max_length=32),
        ),
        migrations.AlterField(
            model_name='productimage',
            name='image',

field=models.ImageField(upload_to=restapi.models.get_product_gallery_upload_location),
        ),
        migrations.CreateModel(
            name='Order',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
                ('delivery_address', models.TextField(blank=True, null=True)),
                ('status', models.CharField(choices=[('AWAITING CONFIRMATION',
'AWAITING_CONFIRMATION'), ('PAYED', 'PAYED'), ('CONFIRMED', 'CONFIRMED'), ('IN
PROCESSING', 'IN_PROCESSING'), ('SEND', 'SEND'), ('DELIVERED', 'DELIVERED'),
('REJECTED', 'REJECTED'), ('CANCELLED', 'CANCELLED'), ('RETURN', 'RETURN')],
max_length=32)),
                ('ordered_at', models.DateTimeField(auto_now_add=True)),
                ('products', models.ManyToManyField(related_name='orders', to='restapi.product')),
```

```
                ('user', models.ForeignKey(on_delete=django.db.models.deletion.PROTECT,
to=settings.AUTH_USER_MODEL)),
            ],
        ),
    ]
```

0009_productsize.py

```
# Generated by Django 4.2.7 on 2024-01-24 13:32

from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('restapi', '0008_alter_product_department_alter_productimage_image_and_more'),
    ]

    operations = [
        migrations.CreateModel(
            name='ProductSize',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
                ('size', models.IntegerField()),
            ],
        ),
    ]
```

0010_productsize_product.py

```
# Generated by Django 4.2.7 on 2024-01-24 13:41

from django.db import migrations, models
import django.db.models.deletion


class Migration(migrations.Migration):
```

```
        dependencies = [
            ('restapi', '0009_productsize'),
        ]

        operations = [
            migrations.AddField(
                model_name='productsize',
                name='product',
                field=models.ForeignKey(null=True, on_delete=django.db.models.deletion.CASCADE,
related_name='sizes', to='restapi.product', verbose_name='sizes'),
            ),
        ]
```

0011_profile_delete_order.py

```
# Generated by Django 4.2.7 on 2024-03-17 16:20

from django.conf import settings
from django.db import migrations, models
import django.db.models.deletion


class Migration(migrations.Migration):

    dependencies = [
        migrations.swappable_dependency(settings.AUTH_USER_MODEL),
        ('restapi', '0010_productsize_product'),
    ]

    operations = [
        migrations.CreateModel(
            name='Profile',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
                ('city', models.CharField(blank=True, max_length=128, null=True)),
                ('phone_number', models.CharField(blank=True, max_length=12, null=True)),
                ('user', models.OneToOneField(on_delete=django.db.models.deletion.CASCADE,
to=settings.AUTH_USER_MODEL)),
```

```
            ],
        ),
        migrations.DeleteModel(
            name='Order',
        ),
    ]
```