Міністерство освіти і науки України
Західноукраїнський національний університет
Факультет інформаційно-комунікаційних технологій
Кафедра економічної кібернетики та інформатики

**МІЖДИСЦИПЛІНАРНА КУРСОВА РОБОТА**
з спеціальності 124 «Системний аналіз» за першим (бакалаврським) рівнем вищої освіти на тему:
«Інформаційна сторінка інтернет-магазину »

Студента 4 курсу групи СА-41
спеціальності 124 «Системний аналіз»
Вовк П.І.
(прізвище та ініціали) ( підпис)

Керівник
_____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали)

Національна шкала_____
Кількість балів_____Оцінка____
ECTS_____

Члени комісії:_____ _____
(прізвище та ініціали) ( підпис)
_____ _____
(прізвище та ініціали) ( підпис)
_____ _____
(прізвище та ініціали) ( підпис)

м. Тернопіль 2024 рік

Зміст

# Вступ

1.Актуальність теми: чому інтернет-магазини важливі у сучасному світі.

2.Мета і завдання роботи: створення функціонального інтернет-магазину дзеркал.

3.Огляд методів і технологій: використання сучасних інструментів розробки (React, Django тощо).

## 1. Актуальність теми: чому інтернет-магазини важливі у сучасному світі.

У сучасному світі інтернет-магазини стали невід'ємною частиною комерційного середовища та щоденного життя мільйонів людей. Розвиток цифрових технологій і глобальна діджиталізація економіки сприяли зростанню популярності електронної комерції, яка відкриває безліч можливостей для покупців і продавців.

**Зручність для покупців:**

Інтернет-магазини дозволяють купувати товари, не виходячи з дому, що особливо важливо в умовах швидкого ритму життя. Покупці можуть обирати продукцію, порівнювати ціни та отримувати замовлення з доставкою в зручне місце.

**Доступність:**

Завдяки інтернет-магазинам, товари доступні не лише у великих містах, а й у віддалених регіонах. Це дає змогу усім користувачам отримувати однаковий доступ до якісної продукції.

**Економія ресурсів:**

Для бізнесу інтернет-магазини є способом скоротити витрати на оренду приміщень, персонал і логістику. Водночас покупці економлять час і кошти, уникаючи фізичного відвідування магазинів.

**Персоналізація покупок:**

Завдяки аналітиці даних і технологіям рекомендацій, інтернет-магазини пропонують індивідуальний підхід до клієнтів. Це підвищує зручність і ефективність шопінгу.

**Виклики пандемії COVID-19:**

Останні роки продемонстрували, наскільки важливими є інтернет-магазини у періоди криз. Під час локдаунів вони забезпечували людей базовими товарами, підтримуючи економічну активність у багатьох галузях.

**Широкий вибір:**

В інтернет-магазинах представлені товари, які можуть бути відсутніми у звичайних магазинах, що дає покупцям ширші можливості для вибору.

**Можливості для бізнесу:**

Для підприємців інтернет-магазини є способом вийти на глобальний ринок. Зокрема, невеликі компанії та стартапи можуть конкурувати з великими гравцями завдяки якісному онлайн-сервісу.

Розробка інтернет-магазину дзеркал, яка поєднує сучасні технології, зручність користування та базовий функціонал, дозволяє задовольнити попит як покупців, так і бізнесу. Такий проєкт демонструє реальну цінність сучасних цифрових рішень і відображає тенденції розвитку електронної комерції.

## 2.Мета роботи:

Розробка функціонального інтернет-магазину дзеркал, який забезпечить зручний доступ до товарів, базовий набір функцій для взаємодії користувачів із системою (перегляд продуктів, аутентифікація, аналітика) та використання сучасних веб-технологій для реалізації швидкого, ефективного та надійного програмного рішення.

**Завдання роботи:**

**Аналіз вимог до інформаційної сторінки магазину дзеркал:**

Визначити основний функціонал системи, включаючи аутентифікацію користувачів, адміністративну панель, та перегляд продуктів.

Сформулювати ролі користувачів: гість, зареєстрований користувач, адміністратор.

**Проектування архітектури програми:**

Розробити архітектуру інтернет-магазину, яка передбачає інтеграцію фронтенд- і бекенд-частин через REST API.

Продумати структуру бази даних для зберігання інформації про користувачів, продукти та аналітику.

**Розробка фронтенду:**

Використати React для створення інтерфейсу користувача.

Реалізувати адаптивний дизайн за допомогою Tailwind CSS.

Інтегрувати Chart.js для візуалізації аналітичних даних.

**Розробка бекенду:**

Використати Django для серверної частини програми.

Налаштувати Django Rest Framework для створення REST API.

Реалізувати JWT-аутентифікацію для безпечного доступу користувачів.

Реалізація функціоналу програми:

Створити головну сторінку з відображенням списку продуктів.

Забезпечити пошук товарів і доступ до детальної інформації про них.

Реалізувати адміністративну панель з можливістю керування товарами та перегляду аналітики продажів.

**Тестування і налагодження системи:**

Провести тестування аутентифікації та роботи користувачів із системою.

Перевірити коректність роботи адмін-панелі та функціональність пошуку продуктів.

**Документація проекту:**

Створити технічну документацію, яка включає опис архітектури, основні модулі програми та інструкції для розгортання системи.

## 3. **Огляд методів і технологій: використання сучасних інструментів розробки**

Для створення інтернет-магазину дзеркал використовувалися сучасні інструменти розробки, які забезпечують ефективну інтеграцію фронтенд- і бекенд-частин, зручність у розробці, а також високу продуктивність програми. Нижче наведено ключові технології, використані у проекті.

### **Фронтенд**

### **React:**

- JavaScript-бібліотека для створення інтерфейсів користувача.
- Забезпечує компонентний підхід до побудови веб-застосунків, що дозволяє розробляти функціональні, реюзабельні та модульні елементи.
- Використовується для динамічного відображення даних, взаємодії з REST API, а також для побудови адаптивного і зручного інтерфейсу користувача.

### **Tailwind CSS**

- Утилітарний CSS-фреймворк, що дозволяє створювати адаптивний дизайн за допомогою готових класів.
- Забезпечує швидку розробку, мінімалістичний підхід до стилізації та полегшує управління стилями компонентів.
- Використовувався для оформлення інтерфейсу програми, надаючи йому сучасного вигляду.

### **Chart.js**

- Бібліотека для візуалізації даних у вигляді графіків і діаграм. Використовується для відображення аналітики продажів у адміністративній панелі.

- Дозволяє легко створювати різні типи графіків, налаштовуючи їх під потреби проекту.

**Axios**

- Бібліотека для HTTP-запитів, яка забезпечує обмін даними між клієнтською частиною та сервером.
- Використовується для взаємодії з REST API, відправки даних аутентифікації, отримання списків продуктів та інших ресурсів.

**Бекенд**

**1.Django**

- Потужний веб-фреймворк на Python, який забезпечує високу продуктивність і безпеку веб-застосунків.
- Використовується для створення серверної частини програми, управління базою даних і реалізації бізнес-логіки.

**2.Django Rest Framework (DRF)**

- Інструмент для створення REST API на базі Django.
- Спрощує створення ендпоінтів для обміну даними між клієнтом і сервером.
- Забезпечує можливості для побудови CRUD-функціоналу (створення, читання, оновлення, видалення) для роботи з продуктами, користувачами та аналітикою.

**3.JWT (JSON Web Tokens)**

- Технологія для аутентифікації користувачів.
- Дозволяє генерувати токени для авторизації, забезпечуючи безпечний доступ до приватних ресурсів.

- Використовується для розмежування ролей (гостьовий доступ, користувач, адміністратор).

**База даних**

SQLite (вбудована база даних у Django)

- Зберігає інформацію про користувачів, продукти, а також дані аналітики
- Легка у використанні під час розробки, може бути замінена на більш масштабоване рішення (наприклад, PostgreSQL) для реального застосування.

**Інтеграція технологій**

1.Архітектура клієнт-сервер:

- Клієнтська частина (React) виконує запити до бекенда (Django) через REST API, обробляє отримані дані та відображає їх користувачеві.
- Взаємодія забезпечується через Axios і стандарти HTTP.

2.Модульність і реюзабельність:

- Фронтенд побудовано з використанням компонентного підходу, що дозволяє легко розширювати функціонал.
- Бекенд базується на DRF, що дозволяє швидко додавати нові ендпоінти або розширювати базу даних.

**Переваги вибраних технологій**

1.Сучасність: Усі інструменти є актуальними у сфері розробки веб-застосунків.

2.Гнучкість: Легко адаптуються до змін вимог проекту.

3.Продуктивність: Забезпечують високу швидкість роботи застосунку і легкість у підтримці.

4.Відкритий код: React, Django, Tailwind CSS та інші є open-source інструментами, що сприяє широкій підтримці розробниками.

Застосування цих технологій дозволило створити функціональний інтернет-магазин, який відповідає сучасним стандартам розробки.

У цьому розділі розглянуто мету проекту, функціональні вимоги, обмеження та проблеми, а також ролі користувачів системи. Детальний аналіз допоможе зрозуміти концепцію та ключові аспекти створення інтернет-магазину дзеркал.

## 1.1. Мета проекту

- Метою проекту є створення сучасного інтернет-магазину дзеркал, який:
- Забезпечить зручний доступ до каталогу продукції для користувачів із різними рівнями прав доступу.
- Реалізує базові можливості електронної комерції, такі як перегляд товарів, пошук і управління продуктами.
- Надасть адміністраторам інструменти для управління товарами та аналізу популярності продукції.
- Використовуватиме сучасні технології для забезпечення інтерактивного інтерфейсу, швидкої роботи та безпеки даних.

Проект спрямований на вирішення реальних завдань малого бізнесу, демонструючи можливості цифровізації торгівлі через просте, інтуїтивне рішення.

## 1.2. Вимоги до функціоналу

Інтернет-магазин має включати базові функції, необхідні для забезпечення роботи як із боку користувачів, так і з боку адміністратора.

Характерною рисою інтернет-магазину є повна автоматизація системи обробки замовлень, завдяки чому можна працювати індивідуально з кожним зареєстрованим клієнтом. Спільною рисою для та інтернет-магазину та ТІС є можливість здійснювати повний торговий цикл у режимі підключення до мережі. При цьому ТІС додатково інтегрована в систему внутрішнього документообігу компанії. Неавтоматизованими для інтернет-магазину та ТІС

залишаються системи доставки товару. Приблизна структура основних форм організації інтернет-магазинів представлена на рис.3.2


Рисунок 3.2 – Форми організації інтернет-магазинів в українському сегменті мережі Інтернет

### 1.2.1. Базовий функціонал для всіх користувачів (гостьовий доступ):

➢ Перегляд каталогу товарів:

- Відображення списку продуктів із фотографіями, назвою, ціною та описом.
- Реалізація інтуїтивної навігації, яка дозволяє швидко знайти потрібний розділ сайту.

➢ Адаптивний дизайн:

- Інтерфейс має бути зручним для використання як на комп’ютері, так і на мобільних пристроях.

### 1.2.2. Функціонал для зареєстрованих користувачів:

- Реєстрація та авторизація:

- Можливість створити обліковий запис, вказуючи базові дані (ім’я користувача, пароль).
- Реалізація входу в систему через введення логіна і пароля.

- Відображення профілю:

- Після входу користувач бачить свій нікнейм у верхній частині сторінки.

- Вихід із системи:

- Користувач має змогу безпечно завершити сеанс роботи через функцію "Вихід".

### 1.2.3. Функціонал для адміністратора:

- Доступ до адміністративної панелі:

- Адмін-панель має вкладки "Products" (для роботи з товарами) та "Analytics" (для перегляду аналітики).

- Керування товарами:

- Додавання нових продуктів із зазначенням характеристик (назва, опис, ціна, зображення).
- Редагування інформації про існуючі товари.
- Видалення продуктів із бази даних.
- Пошук товарів у каталозі (з можливими обмеженнями).

➢ Аналітика:

- Візуалізація даних продажів через графіки та діаграми (популярність товарів, динаміка продажів).
- Відображення ключових метрик для оцінки ефективності продажів.

### 1.2.4. Технічні вимоги:

- Система повинна забезпечувати швидку взаємодію між фронтендом і бекендом через REST API.
- Для аутентифікації користувачів використовується JWT (JSON Web Tokens).
- Реалізація адаптивного дизайну для забезпечення зручності роботи на різних пристроях.

## 1.3. Обмеження та проблеми

1.3.1. Обмеження:

- Додавання нових продуктів можливе лише через адміністративну панель сервера (доступно лише суперкористувачу).
- Створення нових облікових записів адміністраторів здійснюється вручну через базу даних, що ускладнює масштабування.

1.3.2. Проблеми:

➢ Нестабільна аутентифікація:

- При перезавантаженні сторінки токен аутентифікації може зникати, що змушує користувачів повторно вводити дані для входу.
- Ця проблема особливо критична для збереження сесій зареєстрованих користувачів.

- Обмежений пошук продуктів у каталозі:

- Наявні технічні труднощі у реалізації точного пошуку, через що функція працює не завжди коректно.

- Мінімальна автоматизація адміністративних функцій:

- Відсутність інструментів для автоматичного додавання або масового імпорту продуктів.

Попри зазначені обмеження, система здатна виконувати основні функції інтернет-магазину.

### 1.4. Опис ролей користувачів

- 1.4.1. Гість:

- Має доступ до списку товарів і базової інформації про них (назва, ціна, опис).
- Може переглядати сайт без реєстрації та авторизації.

- 1.4.2. Зареєстрований користувач:

- Має доступ до функціоналу гостя.
- Може авторизуватися в системі, використовуючи логін і пароль.
- Після входу бачить свій нікнейм у верхньому меню.
- Може вийти зі свого облікового запису через функцію "Logout".

- 1.4.3. Адміністратор:
- Має всі права, доступні зареєстрованому користувачу.
- Доступ до адміністративної панелі, що дозволяє:
- Керувати продуктами (додавати, редагувати, видаляти).

- Використовувати пошук для швидкого знаходження товарів.
- Аналізувати популярність продукції через вкладку "Analytics".

➢ Може створювати інших адміністраторів вручну через базу даних, але це обмежено виключно роллю суперкористувача.

Такий розподіл ролей забезпечує гнучкість системи, розмежування доступу до її функцій і безпеку даних.

У розробці інтернет-магазину дзеркал було використано набір сучасних технологій, які забезпечили створення функціонального та масштабованого програмного забезпечення. Технології згруповано за фронтенд- і бекенд-частинами.

## 2.1. Фронтенд

Фронтенд – це клієнтська частина веб-застосунку, яка забезпечує взаємодію користувача з системою через інтуїтивний і функціональний інтерфейс. У проекті інтернет-магазину дзеркал реалізація фронтенду ґрунтується на сучасних веб-технологіях, таких як React, Tailwind CSS та Chart.js, що забезпечує швидкодію, адаптивність і зручність у використанні.

**Ключові завдання фронтенду**

1.Відображення інформації:

- Список товарів (дзеркал) із базовою інформацією (назва, ціна, опис, зображення).
- Реалізація інтуїтивного інтерфейсу, що дозволяє користувачам легко знаходити необхідні товари.

2.Аутентифікація користувачів:

- Надання можливості входу та реєстрації для користувачів.
- Відображення персоналізованого інтерфейсу для зареєстрованих користувачів (наприклад, привітання із зазначенням імені).

3.Адміністративна панель:

- Інтерфейс для адміністраторів із функціями керування товарами (додавання, редагування, видалення).
- Розділ аналітики з відображенням даних продажів у вигляді графіків.

4.Взаємодія з сервером:

- Фронтенд виконує запити до бекенду через REST API для отримання та передачі даних (списки товарів, авторизація, аналітика).

У цьому проекті використано такі технології:

**1.React**

- Бібліотека JavaScript для створення динамічного інтерфейсу користувача.
- Забезпечує компонентний підхід до побудови веб-застосунків, що спрощує розробку, налагодження і реюзабельність коду.
- Реалізовано головну сторінку для перегляду списку товарів, форму авторизації, а також адміністративну панель із вкладками для роботи з продуктами та аналітикою.
-

**2.Tailwind CSS**

- CSS-фреймворк, що забезпечує швидку розробку адаптивного дизайну.
- Дозволяє використовувати готові класи для стилізації компонентів без потреби створювати власні стилі.
- Використано для створення привабливого та сучасного дизайну інтерфейсу.

**3.Chart.js**

- Інструмент для візуалізації даних у вигляді графіків.
- Забезпечує створення різних типів діаграм, налаштовуючи їх для аналітики продажів.
- Використовується у вкладці "Analytics" для відображення інформації про популярність продуктів.

**4.Axios**

- Бібліотека для виконання HTTP-запитів з клієнта до сервера.
- Використовується для взаємодії з API, надсилання запитів для аутентифікації, отримання списків продуктів і статистичних даних.

Фронтенд інтернет-магазину дзеркал створено із застосуванням сучасних веб-технологій, що забезпечує швидкодію, зручність використання та

адаптивність. Реалізований інтерфейс є модульним і легко масштабованим, що дозволяє розширювати функціональність проекту відповідно до вимог.

### 2.2. Бекенд

Бекенд – це серверна частина веб-застосунку або програми, яка відповідає за:

- обробку даних,
- виконання бізнес-логіки,
- збереження та управління інформацією в базах даних,
- забезпечення взаємодії з клієнтською частиною (фронтендом).

Іншими словами, бекенд – це «те, що приховано від користувача» і виконує всі внутрішні операції для підтримки функціонування програми.

**Основні функції бекенду**

1.Обробка запитів:

- Коли користувач на фронтенді (наприклад, у браузері) виконує якусь дію (наприклад, перегляд списку товарів), бекенд приймає запит, обробляє його і повертає потрібну відповідь.

2.Управління даними:

- Бекенд відповідає за збереження даних у базі даних, їх оновлення, видалення або передачу фронтенду у потрібному форматі.

3.Виконання бізнес-логіки:

- Це правила та функції, які визначають, як працює програма. Наприклад, обчислення знижок, перевірка прав доступу або автоматизація певних процесів.

4.Аутентифікація та авторизація:

- Бекенд забезпечує безпеку програми, перевіряючи, хто користувач (аутентифікація) і що йому дозволено робити (авторизація).

5.Взаємодія з іншими сервісами:

- Бекенд може надсилати запити до інших програм або сервісів, наприклад, платіжних систем, поштових сервісів або аналітичних платформ.

**Різниця між бекендом і фронтендом**

| Фронтенд | Бекенд |
|---|---|
| Відповідає за інтерфейс користувача. | Відповідає за обробку даних. |
| Реалізується у браузері. | Працює на сервері. |
| Мови: HTML, CSS, JavaScript. | Мови: Python, JavaScript, PHP. |
| Бібліотеки: React, Angular. | Фреймворки: Django, Laravel. |

**Основні компоненти бекенду**

1.Сервер:

- Це програма або апаратне забезпечення, яке приймає та обробляє запити від клієнтів (наприклад, HTTP-запити).

2.База даних:

- Сховище інформації, де зберігаються всі дані програми (наприклад, дані про користувачів, товари, замовлення).
- Приклади: MySQL, PostgreSQL, MongoDB.

3.Серверна логіка:

- Це код, який виконує всі операції, пов’язані з обробкою даних, виконанням бізнес-логіки та взаємодією з базою даних.

4.API (Application Programming Interface):

- Це інтерфейс, який дозволяє фронтенду та бекенду обмінюватися даними. Наприклад, через REST API або GraphQL.

У проекті використано такі інструменти:

**1.Django**

- Фреймворк на Python для розробки веб-застосунків.
- Забезпечує високу продуктивність, зручність роботи з базою даних та надійність системи.

- Реалізовано REST API, що дозволяє клієнту отримувати дані про товари, користувачів та аналітику.

Django – це універсальний фреймворк для розробки веб-додатків, який ідеально підходить для проектів будь-якої складності. Завдяки своїй надійності, швидкості розробки та безпеці, він став одним із найпопулярніших інструментів у сфері веб-програмування.

**2.Django Rest Framework (DRF)**

- Потужний інструмент для створення REST API.
- Спрощує реалізацію CRUD-операцій (створення, читання, оновлення, видалення) для роботи з продуктами та користувачами.
- Дозволяє інтегрувати бекенд із фронтендом через HTTP-запити.

**Використання Django**

1.Розробка веб-сайтів:

Django ідеально підходить для створення динамічних веб-сайтів, корпоративних порталів або блогів.

2.Електронна комерція:

Фреймворк дозволяє створювати інтернет-магазини з інтеграцією платежів, аналітики та керування товарами.

3.REST API:

Django часто використовується для розробки серверної частини веб-додатків із використанням REST API (через Django Rest Framework).

4.Проекти з великим обсягом даних:

Масштабованість і надійність дозволяють використовувати Django для великих проектів, таких як соціальні мережі чи системи управління контентом.

**3.JWT (JSON Web Tokens)**

- Технологія для аутентифікації та авторизації користувачів.
- Використовується для забезпечення безпеки доступу до приватних ресурсів і розмежування ролей (гість, користувач, адміністратор).
- Реалізовано генерацію токенів для авторизації користувачів.

JWT (JSON Web Tokens) — це компактний формат для передачі інформації між сторонами у вигляді JSON-об'єктів, який забезпечує безпечний і перевіряємий спосіб аутентифікації користувачів та обміну даними. JWT часто використовується для авторизації у веб-додатках, мобільних додатках та API.

### 2.3. База даних

**1.SQLite**

- Легка реляційна база даних, вбудована у Django.
- Використовується для зберігання інформації про користувачів, продукти, дані аналітики та авторизації.
- Простота налаштування та інтеграції робить її зручним рішенням на етапі розробки.

### 2.4. Інтеграція технологій

Фронтенд і бекенд взаємодіють за допомогою REST API, що забезпечує обмін даними між клієнтською та серверною частинами.

**Основні аспекти інтеграції:**

- HTTP-запити реалізовано за допомогою Axios.
- Відповіді з бекенду передаються у вигляді JSON, що спрощує обробку даних на фронтенді.
- Токени JWT використовуються для перевірки прав доступу до певних ресурсів.

**Переваги використаних технологій:**

1.Сучасність: Усі технології відповідають сучасним стандартам розробки веб-застосунків.

2.Продуктивність: Забезпечують швидке виконання запитів і відображення даних.

3.Гнучкість: Легко адаптуються до змін у вимогах проекту.

4.Відкритий код: Використання open-source технологій забезпечує доступність великої кількості документації та прикладів.

Завдяки застосуванню цих технологій створено функціональний інтернет-магазин, який поєднує ефективність роботи з сучасним дизайном та гнучкістю у налаштуванні.

# Розділ 3.Опис програмного коду

Цей розділ детально висвітлює технічні аспекти реалізації інтернет-магазину дзеркал. Проект складається з двох основних частин: фронтенду (React) і бекенду (Django). Розглянуто ключові компоненти програми, їх функції, архітектуру та взаємодію.

## 3.1. Фронтенд: Реалізація інтерфейсу користувача

Фронтенд реалізовано на основі React, що забезпечує динамічний, компонентний підхід до створення інтерфейсу.

Основні модулі фронтенду:

1.Структура компонентів:

- App.js: Головний компонент, який відповідає за маршрутизацію та загальну структуру сторінок.
- Header: Відображає навігацію та стан користувача (гостьовий режим, зареєстрований користувач, адміністратор).
- ProductList: Компонент для відображення списку продуктів, отриманих із бекенду через REST API.
- AdminPanel: Реалізує адміністративну панель із вкладками Products (керування товарами) та Analytics (аналітика).
- LoginForm/RegisterForm: Компоненти для аутентифікації та реєстрації користувачів.

2.Основні функції фронтенду:

- Axios-запити:

Функція fetchProducts отримує список продуктів із бекенду через REST API.

```
Javascript
const fetchProducts = async () => {
   const response = await axios.get('/api/products/');
   setProducts(response.data);
};
```

- Обробка аутентифікації:

  Використання JWT-токенів для авторизації користувачів. Після успішного логіну токен зберігається у localStorage:

  javascript

```
const login = async (credentials) => {
   const response = await axios.post('/api/login/', credentials);
   localStorage.setItem('token', response.data.token);
};
```

  3.Використання Tailwind CSS:

- Tailwind CSS використано для стилізації компонентів.
- Приклад класів для адаптивного дизайну кнопки:

  html

```
<button className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded">
   Login
</button>
```

  4.Аналітика (Chart.js):

- Для вкладки Analytics у адмін-панелі реалізовано відображення графіків популярності товарів.
- Приклад використання Chart.js:

  javascript

```
const data = {
   labels: ['Product 1', 'Product 2', 'Product 3'],
   datasets: [{
      label: 'Sales',
      data: [12, 19, 3],
      backgroundColor: ['rgba(75, 192, 192, 0.2)'],
      borderColor: ['rgba(75, 192, 192, 1)'],
      borderWidth: 1
   }]
};
```

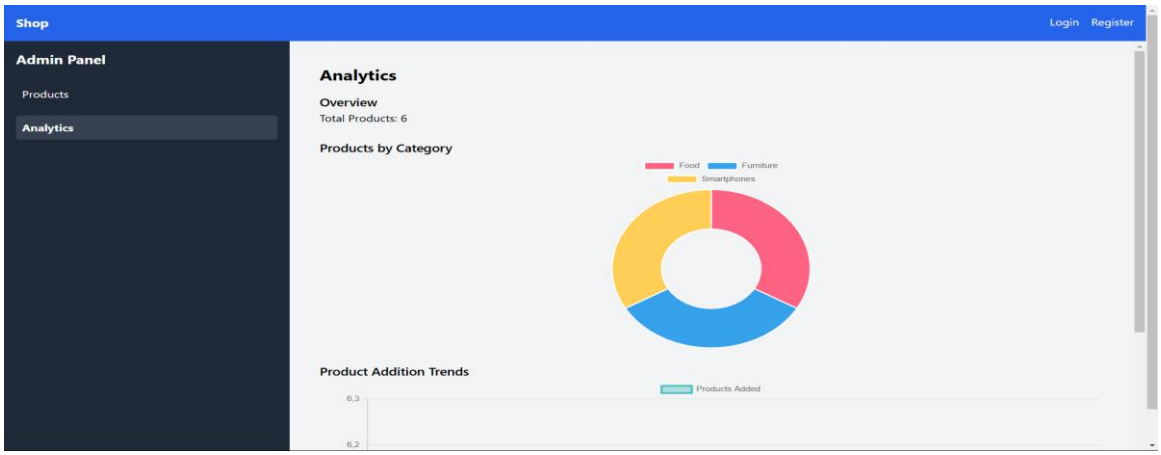
3.2. Бекенд: Реалізація серверної частини

Серверну частину розроблено за допомогою Django і Django Rest Framework (DRF), що забезпечує надійну роботу REST API та взаємодію з базою даних.

1.Основні модулі бекенду:

- **models.py:**

  Містить моделі для зберігання інформації про продукти, користувачів та аналітику.




  python

  class Product(models.Model):

    name = models.CharField(max_length=100)

    description = models.TextField()

    price = models.DecimalField(max_digits=10, decimal_places=2)

    stock = models.IntegerField()

- **serializers.py:**

  Служить для перетворення даних моделей у формат JSON.

  python

  class ProductSerializer(serializers.ModelSerializer):

    class Meta:

      model = Product

      fields = '__all__'

- **views.py:**

  Містить логіку обробки запитів. Наприклад, створення списку продуктів:

**python**

```
class ProductList(APIView):
    def get(self, request):
        products = Product.objects.all()
        serializer = ProductSerializer(products, many=True)
        return Response(serializer.data)
```

**2.Аутентифікація:**

- Для аутентифікації користувачів реалізовано JWT через бібліотеку djangorestframework-simplejwt.
- Налаштування JWT:

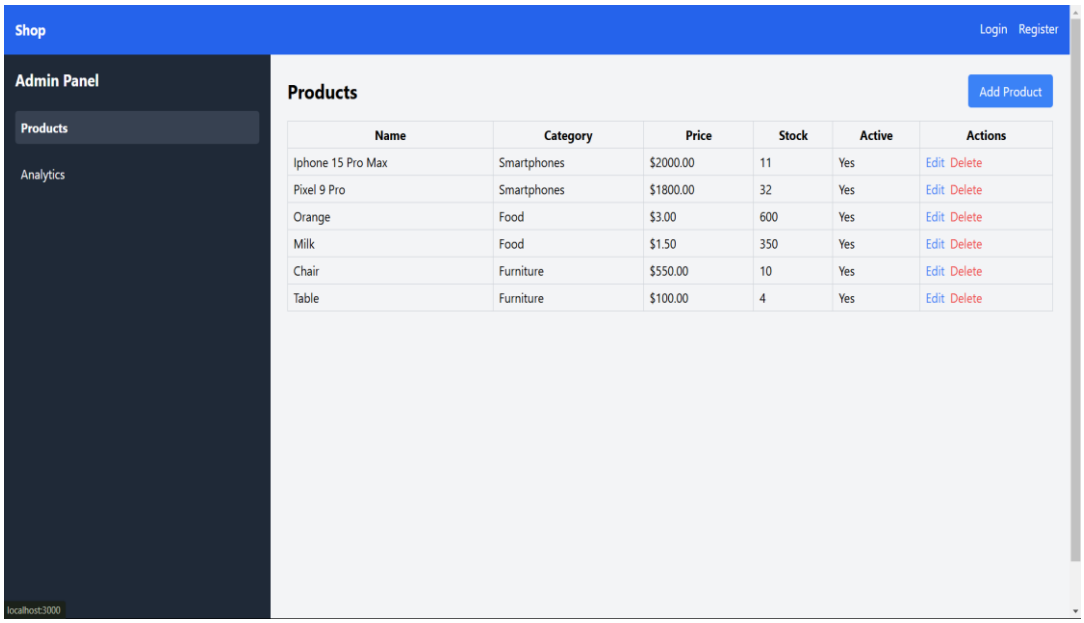
```
python
SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=30),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
    'AUTH_HEADER_TYPES': ('Bearer',),
}
```

- Приклад видачі токена:

```
python
from rest_framework_simplejwt.views import TokenObtainPairView
urlpatterns = [
    path('api/login/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
]
```

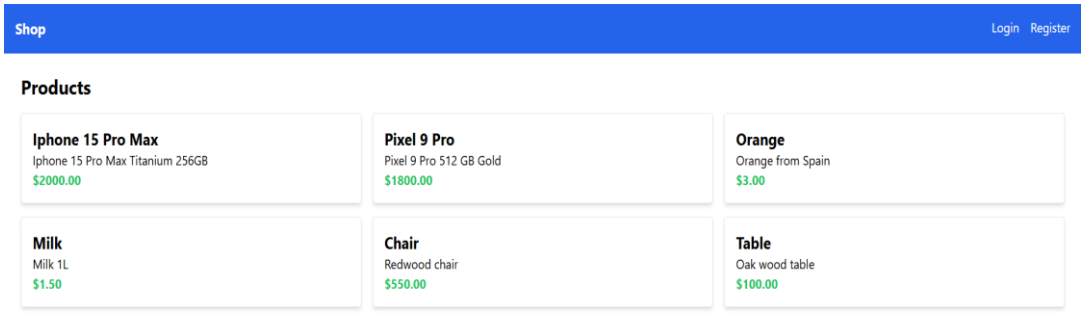
**3.Адмінка Django:**

- Використовується для керування продуктами на сервері (додавання, редагування, видалення).
- Доступ надається суперкористувачу.




**4.Аналітика:**

- У бекенді реалізовано API для збору даних про популярність товарів, які відображаються через фронтенд.




---

### 3.3. Інтеграція клієнтської та серверної частин

1.REST API:

- Фронтенд надсилає запити до бекенду за допомогою HTTP.
- Бекенд відповідає у форматі JSON, який обробляється React-компонентами.

2.Захист даних:

- Для передачі авторизаційних токенів використовується заголовок Authorization: Bearer <token> у запитах.
- Захист приватних маршрутів у фронтенді:

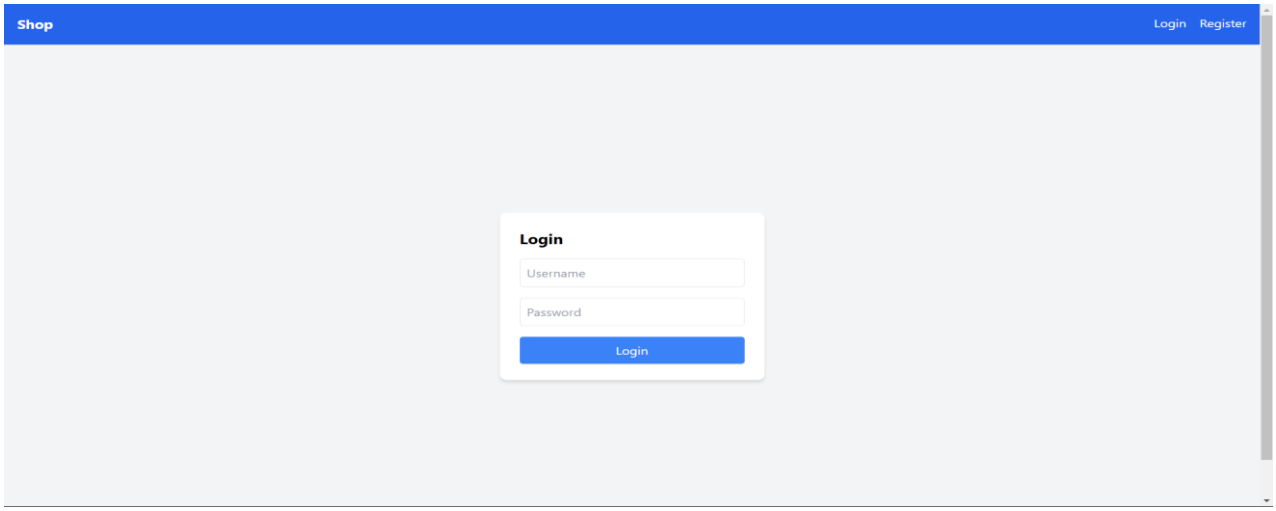
  javascript

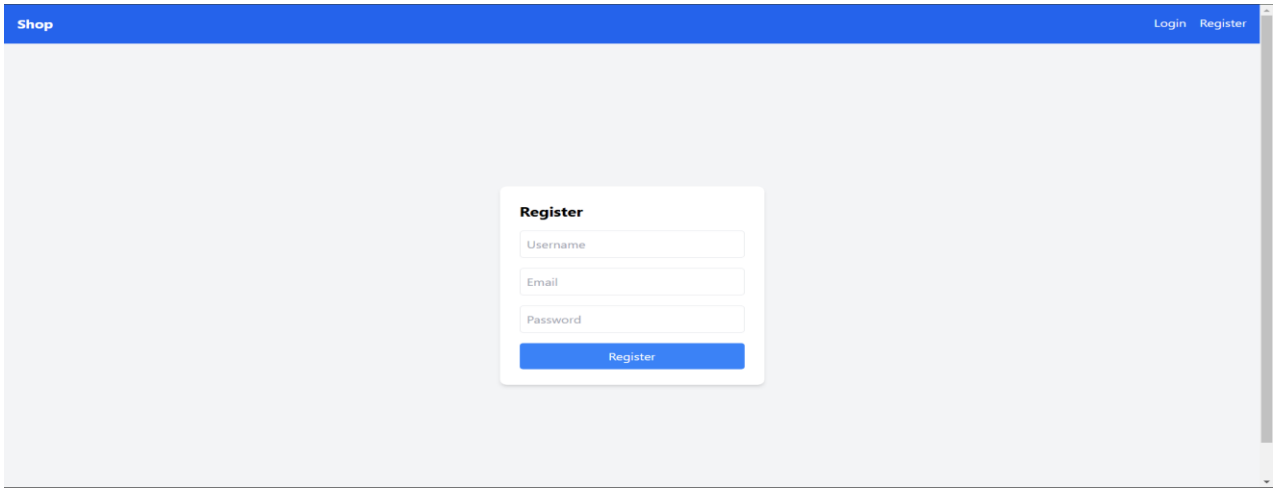
  const PrivateRoute = ({ children }) => {

  const token = localStorage.getItem('token');

  return token ? children : <Navigate to="/login" />;

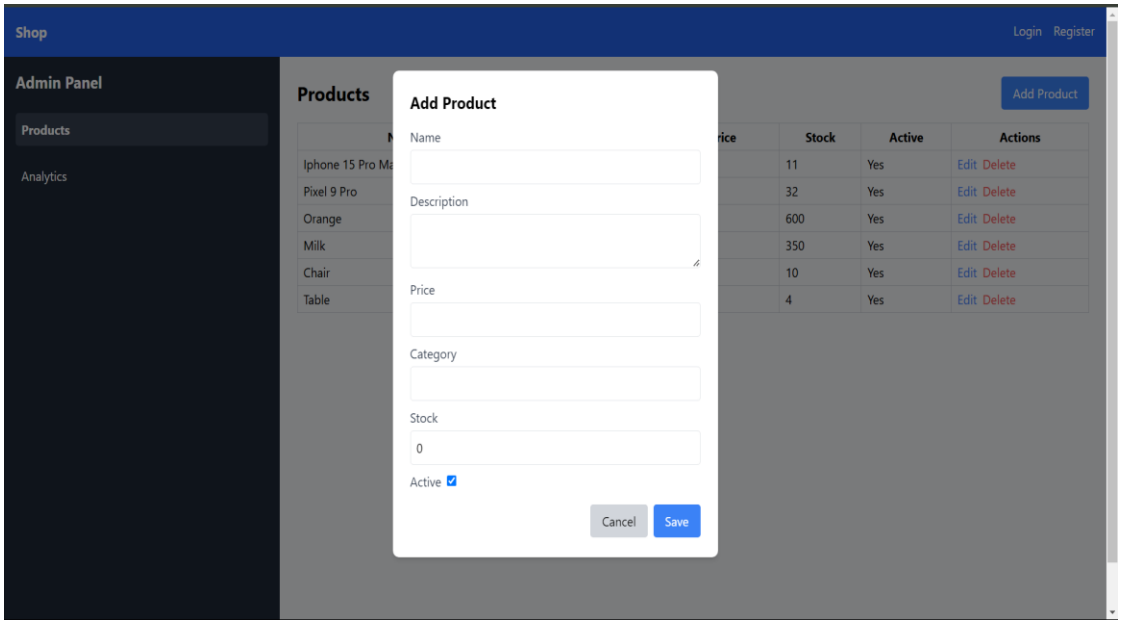
  };



3.Тестування:

- Функціональність тестувалася за допомогою Postman (запити до REST API) і компонентного тестування React.




Реалізований код проекту демонструє інтеграцію сучасних веб-технологій для створення функціонального інтернет-магазину. Використання React, Django та REST API забезпечило зручність розробки та модульність програми, що робить її легкою для масштабування та вдосконалення.

# Висновок

У ході виконання курсової роботи було розроблено інформаційну сторінку інтернет-магазину дзеркал, який реалізує базові функції електронної комерції, такі як перегляд каталогу товарів, керування продуктами в адміністративній панелі, а також аналітика продажів. Проект створено з використанням сучасних технологій, що забезпечують адаптивність, продуктивність і безпеку.

Підсумки виконаної роботи

**Досягнуті результати:**

Створено клієнтську частину за допомогою React із використанням Tailwind CSS для адаптивного дизайну.

Розроблено серверну частину на базі Django та Django Rest Framework із забезпеченням взаємодії через REST API.

Реалізовано три рівні доступу до системи: гість, зареєстрований користувач і адміністратор.

Налаштовано аутентифікацію користувачів за допомогою JWT для захисту даних і контролю доступу.

В адміністративній панелі додано функціонал для роботи з товарами (додавання, редагування, видалення) та перегляду аналітики через графіки (Chart.js).

**Невирішені проблеми**:

Аутентифікація потребує вдосконалення: після перезавантаження сторінки токен інколи втрачається.

Пошук у каталозі товарів має обмежену функціональність і може працювати некоректно.

Додавання нових адміністраторів можливе лише через серверну частину (робота з базою даних).

Значення проекту

Ця курсова робота демонструє здатність застосовувати сучасні інструменти для розробки веб-додатків, враховуючи практичні вимоги бізнесу. Інтернет-

магазин дзеркал може слугувати базою для подальшого вдосконалення і розширення функціональності. Його переваги полягають у використанні надійних фреймворків, модульної архітектури та забезпеченні мінімально необхідного функціоналу для електронної комерції.

Перспективи розвитку проекту

Технічні вдосконалення:

- Оптимізація аутентифікації для забезпечення стабільності токенів (збереження в cookies із безпечними налаштуваннями).
- Розширення можливостей пошуку та фільтрації товарів.
- Додавання нового функціоналу:
- Розробка кошика покупок і системи оформлення замовлень.
- Інтеграція платіжних систем для онлайн-оплат.

Масштабування:

- Перехід на більш потужну базу даних (наприклад, PostgreSQL) для роботи з великим обсягом даних.
- Оптимізація серверної частини для роботи в умовах підвищеного навантаження.
- Автоматизація адміністративних завдань:

Додавання інтерфейсу для створення нових адміністраторів і масового імпорту товарів (наприклад, через файли CSV).

Загальний висновок

Створений інтернет-магазин відповідає цілям, поставленим у ході курсової роботи. Він демонструє базові можливості системи електронної комерції та використання сучасних веб-технологій. Ця робота не лише є корисною в навчальному контексті, але й закладає основу для подальшого вдосконалення проекту в реальних умовах.

Перелік використаної літератури

1.Бенкс К., Моффат Дж. Вивчення React: сучасні патерни для розробки додатків React. O'Reilly Media, 2023.

2. Django Software Foundation. Офіційна книга Django. Доступно онлайн: https://djangobook.com.

3. Сторі Н. Розробка REST API з Django. Packt Publishing, 2020.

4. "Побудова REST API з Django Rest Framework", блог Dev.to: https://dev.to.

5. "Використання JWT у веб-додатках", блог Auth0: https://auth0.com/blog.

6. "Кращі практики роботи з Django", блог RealPython: https://realpython.com.

7. Django Rest Framework - Веб-сайт URL: https://www.django-rest-framework.org/

8. https://dzerkaloreflect.shop/ua/ - Інтернет магазин дзеркал

**Views.py**

```
from django.contrib.auth.hashers import make_password
from django.db.models import Count
from django.db.models.functions import TruncMonth, TruncDay
from django.db import models

from rest_framework import viewsets, permissions, status
from rest_framework.views import APIView
from rest_framework.response import Response

from .models import Product, User
from .serializers import ProductSerializer, UserSerializer



class IsAdminOrReadOnly(permissions.BasePermission):
    """
    Користувач має доступ лише до перегляду (GET),
    а адмін може створювати, оновлювати та видаляти продукти.
    """

    def has_permission(self, request, view):
        if request.method in permissions.SAFE_METHODS:  # GET, HEAD, OPTIONS
            return True
        return request.user.is_authenticated and request.user.is_admin



class ProductViewSet(viewsets.ModelViewSet):
    queryset = Product.objects.all()
    serializer_class = ProductSerializer
```

```
    permission_classes = [IsAdminOrReadOnly]


class RegisterUserView(APIView):
    def post(self, request):
        data = request.data
        if data.get("is_admin") is True and not request.user.is_authenticated:
            return Response({"detail": "Only an admin can create another admin."},
status=status.HTTP_403_FORBIDDEN)

        try:
            user = User.objects.create(
                username=data['username'],
                email=data['email'],
                password=make_password(data['password']),
                is_admin=data.get("is_admin", False)  # За замовчуванням користувач
не є адміністратором
            )
            serializer = UserSerializer(user)
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        except Exception as e:
            return
Response({"error": str(e)},
 status=status.HTTP_400_BAD_REQUEST)

class UserInfoView(APIView):
    permission_classes = [permissions.IsAuthenticated]

    def get(self, request):
        user = request.user
        return Response({
```

```
            "username": user.username,
            "email": user.email,
            "is_admin": user.is_staff
        })

class AnalyticsView(APIView):
    permission_classes = [permissions.IsAdminUser]

    def get(self, request):
        total_products = Product.objects.count()
        active_products = Product.objects.filter(is_active=True).count()
        stock_count
Product.objects.aggregate(total_stock=models.Sum('stock'))['total_stock']
        products_by_category
Product.objects.values('category').annotate(count=Count('id'))

        product_trends = (
            Product.objects.annotate(month=TruncMonth('created_at'))
            .values('month')
            .annotate(count=Count('id'))
            .order_by('month')
        )

        response_data = {
            "total_products": total_products,
            "active_products": active_products,
            "total_stock": stock_count,
            "products_by_category": products_by_category,
            "product_trends": list(product_trends),
        }
        return Response(response_data)
```

**models.py**

```
from django.contrib.auth.models import AbstractUser

from django.db import models

class User(AbstractUser):

    is_admin = models.BooleanField(default=False)

class Product(models.Model):

    name = models.CharField(max_length=100)

    description = models.TextField()

    price = models.DecimalField(max_digits=10, decimal_places=2)

    category = models.CharField(max_length=100, blank=True, null=True)  # Для категоризації

    stock = models.PositiveIntegerField(default=0)  # Кількість на складі

    is_active = models.BooleanField(default=True)  # Чи продукт доступний для покупки

    created_at = models.DateTimeField(auto_now_add=True)

    updated_at = models.DateTimeField(auto_now=True)
```

```
 def __str__(self):

        return self.name
```

**admin.py**

```
from django.contrib import admin

from django.contrib.auth.admin import UserAdmin as BaseUserAdmin

from .models import User, Product

class UserAdmin(BaseUserAdmin):

    # Відображення полів у списку користувачів

    list_display = ('username', 'email', 'is_admin', 'is_staff', 'is_superuser')

    list_filter = ('is_admin', 'is_staff', 'is_superuser')

    search_fields = ('username', 'email')

    ordering = ('username',)

# Редагування користувачів

    fieldsets = (

        (None, {'fields': ('username', 'email', 'password')}),
```

```
        ('Permissions', {'fields': ('is_admin', 'is_staff', 'is_superuser', 'groups',
'user_permissions')}),

        ('Important dates', {'fields': ('last_login', 'date_joined')}),

    )

   # Для створення користувачів через адмінку

    add_fieldsets = (

        (None, {

            'classes': ('wide',),

            'fields': ('username', 'email', 'password1', 'password2', 'is_admin', 'is_staff')}


    ),

    )


   filter_horizontal = ('groups', 'user_permissions')

# Реєстрація користувачів і продуктів

admin.site.register(User, UserAdmin)

admin.site.register(Product)
```

**serializers.py**

```
from rest_framework import serializers
from .models import User, Product
class UserSerializer(serializers.ModelSerializer):
 password = serializers.CharField(write_only=True)
  class Meta:
       model = User

        fields = ['id', 'username', 'email', 'password']

    def create(self, validated_data):

        user = User.objects.create_user(

            username=validated_data['username'],

            email=validated_data['email'],

            password=validated_data['password']

        )

        return user

    def get_is_admin(self, obj):

        return obj.is_admin

class ProductSerializer(serializers.ModelSerializer):
```

```
    class Meta:
        model = Product
        fields = '__all__'
```

**urls.py**

```
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import ProductViewSet, RegisterUserView, UserInfoView, AnalyticsView
router = DefaultRouter()
router.register(r'products', ProductViewSet)
urlpatterns = [
    path('', include(router.urls)),
    path('register/', RegisterUserView.as_view(), name='register'),
    path('user-info/', UserInfoView.as_view(), name='user-info'),
    path('analytics/', AnalyticsView.as_view(), name='analytics'),
]
```

## FRONTEND

### App.js

```
import React, { useState } from "react";
import { BrowserRouter as Router, Routes, Route, Navigate } from "react-router-dom";
import Navbar from "./components/Navbar";
import Login from "./pages/Login";
import Register from "./pages/Register";

import Products from "./pages/Products";
import AdminLayout from "./components/admin/AdminLayout";
import AdminAnalytics from "./components/admin/AdminAnalytics";
import AdminProducts from "./components/admin/AdminProducts";

const App = () => {
  const [user, setUser] = useState(null);

  const logout = () => {
    localStorage.removeItem("access");
    localStorage.removeItem("refresh");
    localStorage.setItem("isAuthenticated", false);
    setUser(null);
  };


  return (
    <Router>
      <Navbar user={user} logout={logout} />
      <div className="container">
```

```
        <Routes>
          <Route path="/" element={<Products />} />
          <Route path="/login" element={<Login setUser={setUser} />} />
          <Route path="/register" element={<Register />} />
          <Route
            path="/admin"
            element={user && user.is_admin ? <AdminLayout /> : <Navigate to="/"
/>}
          >
            <Route path="products" element={<AdminProducts />} />
            <Route path="analytics" element={<AdminAnalytics />} />
          </Route>
        </Routes>
      </div>
    </Router>
  );
};

export default App;
```

**utils/api.js**

```
import axios from "axios";

export const API_URL = "http://127.0.0.1:8000/api";

export const authHeader = () => {
  const token = localStorage.getItem("access");
  return token ? { Authorization: `Bearer ${token}` } : {};
};

export const login = async ({ username, password }) => {
```

```
  const response = await axios.post(`${API_URL}/token/`, { username, password
});
  localStorage.setItem("access", response.data.access);
  localStorage.setItem("refresh", response.data.refresh);

  // Отримання інформації про користувача
  const userResponse = await axios.get(`${API_URL}/user-info/`, {
    headers: {
      Authorization: `Bearer ${response.data.access}`,
    },
  });

  return userResponse.data; // Повертаємо інформацію про користувача
};

export const fetchProducts = async () => {
  const response = await axios.get(`${API_URL}/products/`);
  return response.data;
};

export const createProduct = async (product, token) => {
  const response = await axios.post(`${API_URL}/products/`, product, {
    headers: { Authorization: `Bearer ${token}` },
  });
  return response.data;
};

export const getProducts = async ({ search }) => {
  const response = await axios.get(`${API_URL}/products/`, {
    params: { search },
    ...authHeader(),
```

```
  });
  return response.data;
};

export const deleteProduct = async (id) => {
  await axios.delete(`${API_URL}/products/${id}/`, { headers: authHeader() });
};

export const saveProduct = async (id, product) => {
  if (id) {
    return await axios.put(`${API_URL}/products/${id}/`, product, { headers:
authHeader() });
  } else {
    return await axios.post(`${API_URL}/products/`, product, { headers:
authHeader() });
  }
};

export const isAdmin = async (token) => {
  const response = await axios.get(`${API_URL}/user/`, {
    headers: { Authorization: `Bearer ${token}` },
  });

  return response.data;
};

export const refreshAccessToken = async () => {
  const refresh = localStorage.getItem("refresh");
  if (!refresh) return;

  try {
```

```
    const response = await axios.post(`${API_URL}/token/refresh/`, { refresh });
    const { access } = response.data;
    localStorage.setItem("access", access);
    return access;


} catch (err) {
    console.error("Failed to refresh access token", err);
    logout();
  }
};

export const logout = () => {
  localStorage.removeItem("access");
  localStorage.removeItem("refresh");

  localStorage.setItem("isAuthenticated", false);
  window.location.reload();
};

export const fetchAnalytics = async () => {
  const response = await axios.get(`${API_URL}/analytics/`, { headers:
authHeader() });
  return response.data;
};
```

components/Navbar.js

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

const Navbar = ({ user, logout }) => {
```

```
return (
  <nav className="bg-blue-600 text-white p-4">
    <div className="container mx-auto flex justify-between">
      <Link to="/" className="text-lg font-bold">
        Shop
      </Link>
      <div>
        {!!user && localStorage.getItem("isAuthenticated") === "true" ? (
          <>
            <span className="mr-4">Welcome, {user.username}!</span>
            <button onClick={logout} className="text-red-500">
              Logout
            </button>
            {user.is_admin && (
              <Link to="/admin" className="ml-4">
                Admin
              </Link>
            )}
          </>
        ) : (
          <>
            <Link to="/login" className="mr-4">
              Login
            </Link>
            <Link to="/register">Register</Link>
          </>


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

```
  );
};

export default Navbar;
```

**components/admin/AdminAnalytics.js**

```
import React, { useEffect, useState } from "react";
import { fetchAnalytics } from "../../utils/api";
import { Doughnut, Line } from "react-chartjs-2";
import {
  Chart as ChartJS,
  ArcElement,
  Tooltip,
  Legend,
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement,
} from "chart.js";

ChartJS.register(
  ArcElement,
  Tooltip,
  Legend,
  CategoryScale,
  LinearScale,
  PointElement,
  LineElement
);

const AdminAnalytics = () => {
```

```
  const [analytics, setAnalytics] = useState(null);

  useEffect(() => {
    const loadAnalytics = async () => {
      try {
        const data = await fetchAnalytics();

   console.log(data);
        setAnalytics(data);
      } catch (err) {
        console.error("Failed to fetch analytics", err);
        alert("Error loading analytics");
      }
    };

    loadAnalytics();

  }, []);

  if (!analytics) return <div>Loading...</div>;

  const categoryLabels = analytics.products_by_category.map(
    (item) => item.category || "Uncategorized"
  );
  const categoryData = analytics.products_by_category.map((item) => item.count);

  const doughnutData = {
    labels: categoryLabels,
    datasets: [
      {
        data: categoryData,
```

```
      backgroundColor: ["#FF6384", "#36A2EB", "#FFCE56", "#4BC0C0",
"#9966FF"],
    },
  ],
 };

 const trendLabels = analytics.product_trends.map((item) => item.month);
 const trendValues = analytics.product_trends.map((item) => item.count);

 const trendData = {
  labels: trendLabels,
  datasets: [
   {
    label: "Products Added",
    data: trendValues,
    fill: false,
    backgroundColor: "rgba(75,192,192,0.4)",

borderColor: "rgba(75,192,192,1)",
    },
  ],
 };

 return (
  <div className="p-4">
   <h1 className="text-2xl font-bold mb-4">Analytics</h1>
   <div className="mb-6">
    <h2 className="text-lg font-semibold">Overview</h2>
    <p>Total Products: {analytics.total_products}</p>
   </div>
```

```
      <div className="mb-6">
        <h2 className="text-lg font-semibold">Products by Category</h2>
        <div className="w-80 mx-auto">
          <Doughnut data={doughnutData} />
        </div>
      </div>

    <div>
        <h2 className="text-lg font-semibold">Product Addition Trends</h2>
        <Line data={trendData} />
      </div>
    </div>
  );
};

export default AdminAnalytics;
```