1. в папці 12\_08\_2025\_StepByStep відкриваємо термінал GitBash
2. створюємо віртуальне середовище: python -m virtualenv .venv
3. активуємо його: source .venv/Scripts/activate
4. Встановлюємо: pip install Django django-crispy-forms crispy-bootstrap5 django-celery-beat celery redis Pillow

0 Django: Основний фреймворк.

0 django-crispy-forms, crispy-bootstrap5: Для легкого та красивого рендерингу форм з Bootstrap 5.

0 django-celery-beat, celery, redis: Для асинхронних завдань (підписка на новини). Redis буде використовуватися як брокер повідомлень для Celery.

0 Pillow: Для роботи з зображеннями в Django (наприклад, для завантаження зображень товарів).

1. знаходячись в цій папці створюємо проект:a) django-admin startproject shop\_SBS\_12\_08\_25

b) django-admin startproject shop\_SBS\_12\_08\_25.(сьогодні Gemini каже що цей варіант гірший "РАВЛИК-ПАВЛИК!!!)

1. йдемо по варанту а) переходимо в папу проекту: cd shop\_SBS\_12\_08\_25

структура зараз така:

12\_08\_2025\_StepByStep/

¦

+-- .venv/ # Віртуальне середовище

¦

L-- shop\_SBS\_12\_08\_25/ # Папка нашого Django-проекту

+-- manage.py # Утиліта командного рядка Django

L-- shop\_SBS\_12\_08\_25/ # Папка з основними налаштуваннями проекту

+-- \_\_init\_\_.py

+-- asgi.py

+-- settings.py # Файл з налаштуваннями проекту

+-- urls.py # Файл маршрутизації (URL-адрес)

L-- wsgi.py

1. створюємо додаток shop: python manage.py startapp shop

12\_08\_2025\_StepByStep/

¦

+-- .venv/

L-- shop\_SBS\_12\_08\_25/

+-- manage.py

+-- shop/ <-- Наш новий додаток

¦ +-- \_\_init\_\_.py

¦ +-- admin.py

¦ +-- apps.py

¦ +-- migrations/

¦ +-- models.py

¦ +-- tests.py

¦ L-- views.py

¦

L-- shop\_SBS\_12\_08\_25/

+-- \_\_init\_\_.py

+-- asgi.py

+-- settings.py

+-- urls.py

L-- wsgi.py

1. трепер реєструємо його в shop\_SBS\_12\_08\_25/settings.py

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

# власні додатки

'shop'

1. — Створення URL-конфігурації застосунку

Створюю в папці shop файл urls.py з таким вмістом:

from django.urls import path

from .views import home,

urlpatterns = [

path('', home)

1. — Створення функцій відображення у views.py

У файлі shop/views.py додаємо :

from django.http import HttpResponse

def home(request):

return HttpResponse('<h1>THIS SHOP PAGE</h1>')

1. Для перевірки запускаємо сервер:

python manage.py runserver

Якщо все зроблено правильно, ти побачиш повідомлення про успішний запуск сервера.

Відкрий у своєму браузері адресу http://127.0.0.1:8000/

Ти побачиш сторінку з великим написом THIS SHOP PAGE, який ми прописали у функції home

1. 12) створюємо першу міграцію для стоврення таблиць

python manage.py migrate

1. 13)

Щоб отримати доступ до адмін-панелі Django, нам потрібно створити обліковий запис адміністратора

Створення суперкористувача (superuser)

Зупини сервер: Перш ніж запускати нову команду, тобі потрібно зупинити сервер, який зараз працює.

Для цього просто натисни комбінацію клавіш Ctrl + C у терміналі.

Виконай команду: Переконайся, що ти знаходишся у кореневій папці проекту (shop\_SBS\_12\_08\_25) та віртуальне середовище активовано.

Потім введи таку команду: python manage.py createsuperuser

Логін admin

Мейл taras14@i.ua

пароль: admin

погоджуємо зміни і то що свариться на пароль

1. 14)

Так, ти абсолютно правий! Першим кроком є створення файлу шаблону.

Давай зробимо це покроково.

1. Створення HTML-шаблону

Створи папку templates. Вона має знаходитися всередині папки нашого додатку shop.

Шлях: shop\_SBS\_12\_08\_25/shop/templates/

Створи папку shop всередині templates. Це є найкращою практикою, щоб уникнути конфліктів назв файлів, якщо у тебе буде декілька додатків.

Шлях: shop\_SBS\_12\_08\_25/shop/templates/shop/

Створи файл home.html всередині цієї останньої папки.

Шлях: shop\_SBS\_12\_08\_25/shop/templates/shop/home.html

Тепер наша структура папок виглядає так:

...

L-- shop/

+-- migrations/

+-- templates/

¦ L-- shop/

¦ L-- home.html

+-- \_\_init\_\_.py

+-- admin.py

+-- apps.py

+-- models.py

+-- tests.py

+-- urls.py

L-- views.py

1. 15)

Оновлення views.py

Щоб Django почав відображати цей шаблон, нам потрібно змінити наш файл shop/views.py. Ми замінимо тимчасову відповідь HttpResponse на функцію render,

яка буде відображати наш новий HTML-файл.

Зміни в файлі shop/views.py:

Відкрий файл shop/views.py і заміни його вміст на наступний код:

Python

from django.shortcuts import render

def home(request):

return render(request, 'shop/home.html')

Цей рядок вказує Django, що він має завантажити та відобразити шаблон home.html з папки shop.

Django автоматично шукатиме цей файл у папці templates нашого додатку.

1. 16)

почнемо наповнювати шаблон home.html крок за кроком.

Базова HTML-структура

Спочатку створимо базову розмітку. Відкрий файл shop/templates/shop/home.html і додай наступний код:

HTML

<!DOCTYPE html>

<html lang="uk">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Головна сторінка</title>

</head>

<body>

<h1>Вітаю на головній сторінці інтернет-магазину!</h1>

</body>

</html>

<!DOCTYPE html>: Оголошення типу документа.

<html lang="uk">: Корінь HTML-документа, що вказує на українську мову.

<head>: Містить метаінформацію, яка не відображається на сторінці (кодування, адаптивність, заголовок).

<title>: Текст, який відображається на вкладці браузера.

<body>: Вся видима частина сторінки.

1. 17)Створення базового шаблону

Щоб не дублювати код (наприклад, навігаційну панель або футер) на кожній сторінці, ми створимо базовий шаблон base.html.

Тобто- куди ми би не переходили, в нас постійно буде одинаков сторінка

Створи файл base.html у папці shop/templates/shop/.

Додай до нього наступний код:

<!DOCTYPE html>

<html lang="uk">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>{% block title %}Мій магазин{% endblock %}</title>

</head>

<body>

<main>

{% block content %}

{% endblock %}

</main>

</body>

</html>

--- {% block title %} та {% block content %}: Це спеціальні теги Django. Вони означають, що в ці місця можна буде вставляти вміст з інших шаблонів, які успадковують цей базовий.

1. 18)

Використання базового шаблону в home.html

змінімо наш файл home.html, щоб він успадковував base.html.

Відкрий shop/templates/shop/home.html і заміни його вміст на:

{% extends 'shop/base.html' %}

{% block title %}Головна сторінка{% endblock %}

{% block content %}

<h1>Вітаю на головній сторінці інтернет-магазину!</h1>

{% endblock %}

----

{% extends 'shop/base.html' %}: Цей рядок вказує, що цей шаблон успадковує все від base.html.

{% block ... %}: Тепер ми визначаємо, який вміст буде вставлено у відповідні блоки з base.html.

1. 19)

Додавання навігаційної панелі в base.html

Відкрий файл shop/templates/shop/base.html і додай наступний код у <body> одразу після відкриваючого тегу <body>:

HTML

<body>

<header>

<nav class="navbar navbar-expand-lg navbar-light bg-light">

<a class="navbar-brand" href="/">Мій магазин</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav">

<li class="nav-item active">

<a class="nav-link" href="/">Головна <span class="sr-only">(поточна)</span></a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Каталог</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Кошик</a>

</li>

</ul>

</div>

</nav>

</header>

<main>

{% block content %}

{% endblock %}

</main>

</body>

Ми використали базову розмітку Bootstrap 4 для створення навігаційної панелі. Вона буде добре виглядати на будь-яких пристроях.

Що ми тут зробили?

<header>: Семантичний тег, який позначає шапку сторінки.

<nav>: Контейнер для навігаційних посилань. Класи navbar, navbar-expand-lg та інші — це класи Bootstrap, які стилізують навігацію.

<a class="navbar-brand">: Наш логотип або назва магазину, що веде на головну сторінку.

<button class="navbar-toggler">: Кнопка для мобільних пристроїв, яка розкриває меню.

<ul> та <li> з класом nav-item: Створення списку посилань на різні сторінки (Головна, Каталог, Кошик).

Після збереження файлу, якщо ти запустиш сервер (python manage.py runserver) і оновиш сторінку в браузері, ти побачиш вже більш структуровану сторінку з навігаційною панеллю зверху.

1. 20

Підключення Bootstrap та JavaScript

Відкрий файл shop/templates/shop/base.html і додай наступні рядки в секцію <head>:

HTML

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>{% block title %}Мій магазин{% endblock %}</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

</head>

Це підключить файл стилів Bootstrap. Наша навігаційна панель тепер буде мати гарний вигляд.

Також, для того, щоб кнопка розгортання на мобільних пристроях працювала, нам потрібні скрипти. Додай їх перед закриваючим тегом </body>:

HTML

<main>

{% block content %}

{% endblock %}

</main>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

Пояснення:

Bootstrap CSS: Файл стилів, який робить наші елементи (наприклад, навігаційну панель) красивими.

jQuery, Popper.js, Bootstrap JS: Ці JavaScript-файли потрібні для інтерактивних елементів Bootstrap, таких як випадаючі меню та кнопки.

1. 21

Додавання футера в base.html

Відкрий файл shop/templates/shop/base.html і додай наступний код одразу після закриваючого тегу </main>:

HTML

</main>

<footer class="bg-light text-center text-lg-start mt-auto">

<div class="container p-4">

<div class="row">

<div class="col-lg-6 col-md-12 mb-4 mb-md-0">

<h5 class="text-uppercase">Мій магазин</h5>

<p>

Ми пропонуємо широкий асортимент товарів високої якості.

Завітайте до нас, щоб знайти те, що вам потрібно!

</p>

</div>

<div class="col-lg-6 col-md-12 mb-4 mb-md-0">

<h5 class="text-uppercase">Контакти</h5>

<ul class="list-unstyled mb-0">

<li><a href="#" class="text-dark">Наші контакти</a></li>

<li><a href="#" class="text-dark">Про нас</a></li>

<li><a href="#" class="text-dark">Доставка та оплата</a></li>

</ul>

</div>

</div>

</div>

<div class="text-center p-3" style="background-color: rgba(0, 0, 0, 0.2);">

© 2025 Copyright:

<a class="text-dark" href="/">Мій магазин</a>

</div>

</footer>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

</body>

</html>

Пояснення

<footer class="bg-light ...">: Семантичний тег, який вказує на нижню частину сторінки. Класи Bootstrap bg-light (світло-сірий фон), text-center, mt-auto допомагають його стилізувати.

<div class="container">: Контейнер Bootstrap, який центрує контент та надає йому відступи.

<div class="row">: Створює рядок, всередині якого ми можемо розміщувати стовпці.

<div class="col-lg-6 ...">: Стовпці Bootstrap, які ділять простір на дві частини, щоб розмістити інформацію про магазин та контакти.

Збережи файл і онови сторінку в браузері. Ти побачиш вже повноцінно оформлену сторінку з навігаційною панеллю та футером.

давай відразу поправимо верхню частину : мій магазин(Добавити лого) | полее(кнопка) каталог | поле пошуку товарів | поле(кнопка) "порівняння( або картинка )| " поле(кнопка) "улюблене"

( або картинка сердечко )" | " поле(кнопка) "корзина"( або картинка кошик )" | " поле(кнопка) "особистий кабінет"( або картинка дяпчик )"

Чудова ідея! Це зробить нашу навігаційну панель більш функціональною та професійною. Давайте оновимо наш файл base.html, щоб додати всі ці елементи.

Оновлення base.html

Відкрий файл shop/templates/shop/base.html і внеси наступні зміни.

1. Додавання іконок Font Awesome

Щоб використовувати іконки (сердечко, кошик, чоловічка), нам потрібно підключити бібліотеку Font Awesome. Додай цей рядок у секцію <head> разом з посиланням на Bootstrap CSS.

HTML

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>{% block title %}Мій магазин{% endblock %}</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/css/all.min.css">

</head>

2. Оновлення навігаційної панелі

Заміни весь блок <nav> на наступний. Цей код включає всі елементи, які ти запропонував, і використовує класи Bootstrap для їх розташування та стилізації.

HTML

<body>

<header>

<nav class="navbar navbar-expand-lg navbar-light bg-light">

<a class="navbar-brand" href="/">

<img src="#" alt="Логотип" width="30" height="30" class="d-inline-block align-top">

Мій магазин

</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav mr-auto">

<li class="nav-item">

<a class="nav-link" href="#">Каталог</a>

</li>

</ul>

<form class="form-inline my-2 my-lg-0">

<input class="form-control mr-sm-2" type="search" placeholder="Пошук товарів" aria-label="Search">

<button class="btn btn-outline-success my-2 my-sm-0" type="submit">Пошук</button>

</form>

<ul class="navbar-nav ml-auto">

<li class="nav-item">

<a class="nav-link" href="#" title="Порівняння">

<i class="fas fa-balance-scale"></i>

</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#" title="Улюблене">

<i class="fas fa-heart"></i>

</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#" title="Кошик">

<i class="fas fa-shopping-cart"></i>

</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#" title="Особистий кабінет">

<i class="fas fa-user"></i>

</a>

</li>

</ul>

</div>

</nav>

</header>

</body>

Пояснення змін:

Логотип: Ми додали тег <img> з атрибутом src="#" як заповнювач для майбутнього логотипа.

Каталог: Цей елемент залишається як окреме посилання.

Пошук: Ми створили Bootstrap-форму (form class="form-inline") з полем для введення тексту (<input type="search">) та кнопкою для пошуку.

Іконки: Для "Порівняння", "Улюблене", "Кошик" та "Особистий кабінет" ми використали іконки з Font Awesome. Теги <i class="..."> відображають іконки, а атрибут title додає підказку при наведенні миші.

Збережи файл і онови сторінку в браузері. Тепер твоя навігаційна панель має бути повністю оновлена і містити всі необхідні елементи.

1. ***Додавання динамічного банера в base.html***

**Додай HTML-розмітку для банера**. Відкрий файл shop/templates/shop/base.html. Додай цей код у блок <header> **перед** тегом <nav>.

<header>

<div id="promo-banner" class="text-center py-2 text-white" style="background-color: #3b5998;">

<p id="promo-text" class="m-0"></p>

</div>

<nav class="navbar navbar-expand-lg navbar-light bg-light">

</nav>

</header>

* 

id="promo-banner": Ідентифікатор для нашого банера.

* id="promo-text": Ідентифікатор для тексту, який буде змінюватися.
* style="background-color: #3b5998;": Я встановив синій фон, але ти можеш обрати будь-який інший.

 **Додай JavaScript-код для ротації тексту**. Встав цей код у кінець файлу base.html, **після** всіх скриптів Bootstrap, але **перед** закриваючим тегом </body>.

<script>

const promoTexts = [

"До дня Незалежності знижка 44%!",

"Кожне третє замовлення знижка 5%!",

"Великий розпродаж до свята!"

];

let currentTextIndex = 0;

const promoTextElement = document.getElementById('promo-text');

function updatePromoText() {

promoTextElement.textContent = promoTexts[currentTextIndex];

currentTextIndex = (currentTextIndex + 1) % promoTexts.length;

}

updatePromoText(); // Показати перший текст одразу

setInterval(updatePromoText, 5000); // Змінювати текст кожні 5 секунд

</script>

1. ***19) Оновлення футера в base.html***

Відкрий файл **shop/templates/shop/base.html**. Знайди і заміни весь існуючий блок <footer> на наступний код.

HTML

<footer class="bg-light text-center text-lg-start mt-auto">

<div class="container p-4">

<div class="row">

<div class="col-lg-6 col-md-12 mb-4 mb-md-0">

<p>

Пропонуємо широкий асортимент товарів високої якості.

Завітайте до нас, щоб знайти те, що вам потрібно!

</p>

</div>

<div class="col-lg-6 col-md-12 mb-4 mb-md-0 text-lg-end">

<p class="mb-0">Контактні дані:</p>

<ul class="list-unstyled">

<li><a href="#" class="text-dark">Наші контакти</a></li>

<li><a href="#" class="text-dark">Про нас</a></li>

<li><a href="#" class="text-dark">Доставка та оплата</a></li>

</ul>

</div>

</div>

</div>

<div class="text-center p-3" style="background-color: rgba(0, 0, 0, 0.2);">

© 2025 Copyright:

<a class="text-dark" href="/">Мій магазин</a>

</div>

</footer>

**Пояснення змін:**

* **Структура**: Ми зберегли двоколонкову структуру (<div class="row"> з двома col-lg-6).
* **Текст**: Лівий стовпець тепер містить лише текст-опис.
* **Контакти**: Правий стовпець містить заголовок "Контактні дані" та список посилань. Клас list-unstyled прибирає маркери зі списку, як на твоєму скріншоті. Клас text-lg-end вирівнює контент по правому краю на великих екранах.

1. ***Створення додатку products***

Давай почнемо з першого етапу — створення нового додатку для товарів і категорій.

1. **Відкрий термінал** у кореневій папці твого проєкту (shop\_SBS\_12\_08\_25).
2. **Виконай команду** для створення нового додатку Django:

Bash

python manage.py startapp products

1. **Зареєструй новий додаток**. Відкрий файл shop\_SBS\_12\_08\_25/settings.py і додай products до списку INSTALLED\_APPS.

Python

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'shop',

'products', # Додай цей рядок

]

Після цих кроків у тебе з'явиться нова папка products у структурі проєкту. Тепер ми можемо створювати моделі для товарів і категорій всередині цієї папки. Готовий до цього?

Z Гаразд, продовжуємо!

Ми зупинилися на тому, щоб почати працювати з моделями. Це ключовий етап для створення основи нашого магазину.

21) Створення моделей Category та Product

Відкрий файл products/models.py і додай наступний код:

Python

from django.db import models

from django.urls import reverse

# Модель для категорії товарів.

# Вона буде використовуватися для групування продуктів.

class Category(models.Model):

# 'name' - це назва категорії, наприклад, 'Електроніка' або 'Одяг'.

# db\_index=True створює індекс в базі даних, що прискорює пошук за цим полем.

name = models.CharField(max\_length=200, db\_index=True)

# 'slug' - це коротка мітка для URL, яка складається тільки з букв, цифр,

# дефісів та підкреслень. unique=True гарантує, що slug буде унікальним.

# Наприклад, для 'Електроніка' slug буде 'elektronika'.

slug = models.SlugField(max\_length=200, unique=True)

class Meta:

# ordering=('name',) - сортує категорії за назвою в алфавітному порядку.

ordering = ('name',)

# verbose\_name - це назва моделі в однині, як вона буде відображатися в адмін-панелі.

verbose\_name = 'category'

# verbose\_name\_plural - це назва моделі в множині.

verbose\_name\_plural = 'categories'

# Метод \_\_str\_\_ повертає рядкове представлення об'єкта.

# Він використовується в адмін-панелі та інших місцях для відображення назви об'єкта.

def \_\_str\_\_(self):

return self.name

# Модель для продукту.

# Тут ми описуємо всі характеристики товару.

class Product(models.Model):

# 'category' - це зовнішній ключ (ForeignKey). Він створює зв'язок

# "багато-до-одного" (many-to-one), де багато продуктів належать до однієї категорії.

# related\_name='products' дозволить легко отримувати список товарів для кожної категорії.

# on\_delete=models.CASCADE означає, що якщо категорія буде видалена, то всі

# пов'язані з нею продукти також будуть видалені.

category = models.ForeignKey(Category, related\_name='products', on\_delete=models.CASCADE)

# 'name' - назва товару.

name = models.CharField(max\_length=200, db\_index=True)

# 'slug' - унікальна мітка для URL товару.

slug = models.SlugField(max\_length=200, db\_index=True)

# 'image' - поле для завантаження зображення.

# upload\_to='products/%Y/%m/%d' вказує, що зображення будуть

# зберігатися в папці 'media/products/рік/місяць/день/'.

# blank=True означає, що це поле не є обов'язковим.

image = models.ImageField(upload\_to='products/%Y/%m/%d', blank=True)

# 'description' - текстовий опис товару.

# blank=True дозволяє залишити це поле пустим.

description = models.TextField(blank=True)

# 'price' - ціна товару.

# max\_digits=10 означає, що загальна кількість цифр не може перевищувати 10.

# decimal\_places=2 означає, що після коми буде 2 цифри (для копійок).

price = models.DecimalField(max\_digits=10, decimal\_places=2)

# 'available' - логічне поле (True/False), яке вказує, чи є товар в наявності.

# default=True встановлює значення за замовчуванням.

available = models.BooleanField(default=True)

# 'created' - дата і час створення товару.

# auto\_now\_add=True автоматично встановлює поточну дату і час при створенні об'єкта.

created = models.DateTimeField(auto\_now\_add=True)

# 'updated' - дата і час останнього оновлення товару.

# auto\_now=True автоматично оновлює дату і час при кожному збереженні об'єкта.

updated = models.DateTimeField(auto\_now=True)

class Meta:

# Сортування продуктів за назвою.

ordering = ('name',)

# Замість index\_together використовуй indexes indexes = [ models.Index(fields=['id', 'slug']), ] # Це прискорює запити, які використовують обидва поля.

index\_together = (('id', 'slug'),)

def \_\_str\_\_(self):

return self.name

1. 22) Створення та застосування міграцій

Після того, як ти збережеш ці моделі, потрібно повідомити Django про зміни. Для цього ми використовуємо команди міграції.

1. **Створи файли міграцій**: Відкрий термінал і виконай команду:

Bash

python manage.py makemigrations products

Ця команда створить файл, який описує, як Django має змінити базу даних.

1. **Застосуй міграції**: Виконай команду, щоб застосувати ці зміни до твоєї бази даних:

Bash

python manage.py migrate

1. Створення файла products\_data.json

У кореневій папці твого проекту shop\_SBS\_12\_08\_25 створи новий файл і назви його products\_data.json.

Скопіюй та встав у нього наступний вміст:

[

{

"category": "Електроніка",

"products": [

{

"name": "Смартфон Samsung Galaxy S23",

"price": 32999.00,

"image": "products/images/samsung\_s23.webp",

"description": "Потужний смартфон з камерою професійного рівня та екраном Dynamic AMOLED 2X."

},

{

"name": "Ноутбук Dell XPS 15",

"price": 54999.00,

"image": "products/images/dell\_xps\_15.webp",

"description": "Високопродуктивний ноутбук для творчих завдань з дисплеєм InfinityEdge."

},

{

"name": "Бездротові навушники Sony WH-1000XM5",

"price": 10999.00,

"image": "products/images/sony\_wh1000xm5.webp",

"description": "Флагманські навушники з найкращим у своєму класі шумозаглушенням та чистим звуком."

},

{

"name": "Розумний годинник Apple Watch Series 8",

"price": 15999.00,

"image": "products/images/apple\_watch8.webp",

"description": "Сучасний годинник з функціями моніторингу здоров'я та великим Retina-дисплеєм."

},

{

"name": "Ігрова консоль PlayStation 5",

"price": 24999.00,

"image": "products/images/ps5.webp",

"description": "Консоль нового покоління з надшвидким SSD та підтримкою 4K-геймінгу."

}

]

},

{

"category": "Одяг",

"products": [

{

"name": "Джинси Levis 501 Original",

"price": 3500.00,

"image": "products/images/levis\_501.webp",

"description": "Класичні джинси прямого крою, виготовлені з високоякісного деніму."

},

{

"name": "Футболка чорна, бавовна",

"price": 500.00,

"image": "products/images/black\_tshirt.webp",

"description": "Проста та стильна футболка з натуральної бавовни, універсальна для будь-якого гардеробу."

},

{

"name": "Зимова куртка The North Face",

"price": 12000.00,

"image": "products/images/north\_face\_jacket.webp",

"description": "Тепла куртка, розроблена для екстремальних погодних умов, з водонепроникною тканиною."

}

]

},

{

"category": "Книги",

"products": [

{

"name": "Дюна",

"price": 450.00,

"image": "products/images/duna\_book.webp",

"description": "Епічний науково-фантастичний роман Френка Герберта про боротьбу за контроль над планетою-пустелею."

},

{

"name": "1984",

"price": 380.00,

"image": "products/images/1984\_book.webp",

"description": "Антиутопічний роман Джорджа Орвелла про тоталітарний режим."

},

{

"name": "Володар перснів",

"price": 600.00,

"image": "products/images/lotr\_book.webp",

"description": "Фентезі-епопея Дж. Р. Р. Толкіна про подорож гобіта Фродо для знищення Персня Всевладдя."

},

{

"name": "Гаррі Поттер і філософський камінь",

"price": 420.00,

"image": "products/images/harry\_potter\_book.webp",

"description": "Перша книга із серії романів про хлопчика-чарівника Гаррі Поттера."

},

{

"name": "Автостопом по Галактиці",

"price": 350.00,

"image": "products/images/hitchhikers\_guide.webp",

"description": "Гумористичний науково-фантастичний роман Дугласа Адамса."

}

]

}

]

Z

1. Створив папку 12\_08\_2025\_StepByStep\shop\_SBS\_12\_08\_25\products\images\

Туди закинув 13 картинок по для товарів з файлу(див вище)

1. Створюємо скрипт для завантаження даних в базу

Оскільки ми будемо завантажувати дані з JSON, нам потрібен скрипт. Створи у кореневій папці проєкту (shop\_SBS\_12\_08\_25) новий файл з назвою **import\_data.py** і встав у нього наступний код:

import os

import django

import json

import slugify

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'shop\_SBS\_12\_08\_25.settings')

django.setup()

from products.models import Category, Product

# Завантажуємо дані з JSON-файлу

with open('products\_data.json', 'r', encoding='utf-8') as f:

data = json.load(f)

for category\_data in data:

category\_name = category\_data['category']

# Створюємо slug для категорії

category\_slug = slugify.slugify(category\_name)

# Отримуємо або створюємо категорію

category, created = Category.objects.get\_or\_create(

name=category\_name,

slug=category\_slug

)

print(f"Категорія '{category.name}' {'створена' if created else 'вже існує'}.")

for product\_data in category\_data['products']:

product\_name = product\_data['name']

product\_slug = slugify.slugify(product\_name)

# Перевіряємо, чи існує товар

if not Product.objects.filter(slug=product\_slug).exists():

Product.objects.create(

category=category,

name=product\_name,

slug=product\_slug,

description=product\_data['description'],

price=product\_data['price'],

image=product\_data['image']

)

print(f" Товар '{product\_name}' додано.")

else:

print(f" Товар '{product\_name}' вже існує.")

print("\nЗавантаження даних завершено!")

**Важлива примітка**: Цей скрипт використовує бібліотеку python-slugify, якої у тебе, швидше за все, немає. Її потрібно встановити.

**4. Встановлення python-slugify**

1. Відкрий термінал.
2. Переконайся, що твоє віртуальне середовище активоване.
3. Виконай команду для встановлення бібліотеки:

Bash

pip install python-slugify

Після того, як ти оновиш файл products\_data.json і збережеш його, знову запусти скрипт.

1. **Запусти скрипт import\_data.py** з терміналу:

Bash

python import\_data.py

Цього разу він має успішно завантажити всі товари та категорії в базу даних.

**Перевірка в адмін-панелі**

1. Запусти сервер розробки, якщо він ще не запущений:

Bash

python manage.py runserver

1. Перейди у браузері за адресою: **http://127.0.0.1:8000/admin**
2. Авторизуйся під своїм обліковим записом суперкористувача.

Ти побачиш у списку додатків products з двома моделями: Categories та Products.

* **Categories**: Коли ти перейдеш за цим посиланням, ти побачиш 3 створені категорії: Електроніка, Одяг та Книги.
* **Products**: Тут ти знайдеш 13 товарів, які ми додали, з назвами, цінами та іншою інформацією.

Якщо все відображається правильно, це означає, що імпорт даних пройшов успішно.

**Що далі?**

Тепер, коли в нас є дані, ми можемо переходити до найцікавішого — **створення веб-сторінок** для відображення цих товарів у твоему магазині.

Наступні кроки:

1. Налаштування **URL-адрес** для відображення списку товарів та деталей конкретного товару.
2. Написання **views** (функцій-обробників), які будуть отримувати дані з бази даних.
3. Створення **templates** (HTML-шаблонів) для відображення цих даних у браузері.

Гаразд, чудово! Тепер, коли ми успішно налаштували адмін-панель і завантажили дані, перейдемо до найцікавішої частини — створення фронтенду, тобто сторінок, які бачитимуть користувачі.

Ми створимо:

1. Головну сторінку, де відображатиметься список усіх товарів.
2. Сторінки категорій, де товари будуть відфільтровані.
3. Сторінку деталей товару.

Z

1. Налаштування URL-адрес

Першим кроком буде створення URL-маршрутів у нашому додатку products, щоб Django знав, за якими адресами відображати наші сторінки.

1. У папці **products** створи новий файл і назви його **urls.py**.
2. Додай у цей файл наступний код:

from django.urls import path

from . import views

app\_name = 'products'

urlpatterns = [

path('', views.product\_list, name='product\_list'),

path('<slug:category\_slug>/', views.product\_list, name='product\_list\_by\_category'),

path('<int:id>/<slug:slug>/', views.product\_detail, name='product\_detail'),

]

Z

1. Тепер потрібно підключити ці URL-адреси до головного проєкту. Відкрий файл **shop\_SBS\_12\_08\_25/urls.py** і додай у нього include:

Python

from django.contrib import admin

from django.urls import path, include

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

path('admin/', admin.site.urls),

# Підключаємо URL-адреси з нашого додатку products

path('', include('products.urls', namespace='products')),

]

if settings.DEBUG:

urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

**Пояснення коду**

* app\_name = 'products': Це дозволить нам звертатися до URL-адрес за іменами (products:product\_list), що зручно при роботі з шаблонами.
* path('', views.product\_list, name='product\_list'): Це URL-адреса для головної сторінки (/). Вона буде показувати список усіх товарів.
* path('<slug:category\_slug>/', views.product\_list, name='product\_list\_by\_category'): Це URL-адреса для сторінок категорій (/elektronika/). Вона буде передавати slug категорії у функцію views.product\_list для фільтрації.
* path('<int:id>/<slug:slug>/', views.product\_detail, name='product\_detail'): Це URL-адреса для деталей товару (/1/smartfon-samsung-galaxy-s23/). Вона використовуватиме id та slug товару для його ідентифікації.

Тепер ми готові написати views для обробки цих URL-адрес.

Z

1. 26)Створення Views (логіка додатку)

Гаразд, домовлено. Кожен наступний фрагмент коду буде з детальними коментарями, щоб ти краще розумів, що відбувається в кожній частині. Це допоможе тобі засвоїти матеріал.

**26) Створення Views (логіка додатку)**

Тепер створимо логіку нашого додатку, яка буде отримувати дані з бази даних і передавати їх у шаблони.

1. Відкрий файл **products/views.py**.
2. Заміни його вміст на наступний код з коментарями:

Python

from django.shortcuts import render, get\_object\_or\_404

from .models import Category, Product

# View для відображення списку товарів

# Вона може приймати необов'язковий параметр category\_slug для фільтрації

def product\_list(request, category\_slug=None):

# Спочатку отримуємо всі категорії, щоб відобразити їх на боковій панелі

categories = Category.objects.all()

# Визначаємо початковий запит для товарів: спочатку всі доступні товари

products = Product.objects.filter(available=True)

# Створюємо змінну для поточної категорії

current\_category = None

# Якщо в URL-адресі є slug категорії

if category\_slug:

# get\_object\_or\_404 - це допоміжна функція Django, яка намагається отримати

# об'єкт (у цьому випадку - категорію) за slug.

# Якщо категорія не знайдена, поверне помилку 404.

current\_category = get\_object\_or\_404(Category, slug=category\_slug)

# Фільтруємо товари, щоб відображати лише ті, що належать до поточної категорії

products = products.filter(category=current\_category)

# Формуємо словник context, який буде переданий у шаблон.

# Шаблон зможе отримати доступ до всіх цих даних.

context = {

'current\_category': current\_category,

'categories': categories,

'products': products

}

# render - це функція Django, яка поєднує шаблон з даними (context)

# та повертає готовий HTML-код.

return render(request, 'shop/product/list.html', context)

# View для відображення деталей конкретного товару

def product\_detail(request, id, slug):

# Отримуємо об'єкт Product за двома параметрами: id та slug.

# get\_object\_or\_404 гарантує, що якщо товар не знайдено, буде повернута помилка 404.

product = get\_object\_or\_404(Product, id=id, slug=slug, available=True)

# Передаємо об'єкт товару в шаблон

context = {

'product': product

}

return render(request, 'shop/product/detail.html', context)

**Що далі?**

Тепер, коли ми написали логіку, нам потрібно створити HTML-шаблони, до яких звертаються ці views. Вони відповідають за візуальне відображення даних, які ми отримали з бази даних.

Наступний крок — створення структури папок для шаблонів.

Нам потрібно створити таку структуру папок для шаблонів у додатку products:

products/

├── templates/

│ └── shop/

│ └── product/

│ ├── list.html # Для списку товарів

│ └── detail.html # Для деталей товару

└── ...

**28) Шаблон для списку товарів: list.html**

Створи файл **products/templates/shop/product/list.html** і встав у нього наступний код з коментарями:

HTML

{% extends "base.html" %}

{% load static %}

{% block title %}

{% if current\_category %}

{{ current\_category.name }}

{% else %}

Products

{% endif %}

{% endblock %}

{% block content %}

<h1>

{% if current\_category %}

{{ current\_category.name }}

{% else %}

Products

{% endif %}

</h1>

<div class="row">

<div class="col-md-3">

<h2>Categories</h2>

<ul class="list-group">

<a href="{% url 'products:product\_list' %}" class="list-group-item {% if not current\_category %}active{% endif %}">All</a>

{% for category in categories %}

<a href="{{ category.get\_absolute\_url }}" class="list-group-item {% if current\_category == category %}active{% endif %}">

{{ category.name }}

</a>

{% endfor %}

</ul>

</div>

<div class="col-md-9">

<div class="row">

{% for product in products %}

<div class="col-md-4 mb-4">

<div class="card h-100 text-center">

<a href="{{ product.get\_absolute\_url }}">

<img class="card-img-top" src="{% if product.image %}{% static product.image.url %}{% else %}{% static 'img/no\_image.png' %}{% endif %}" alt="{{ product.name }}">

</a>

<div class="card-body">

<h5 class="card-title">{{ product.name }}</h5>

<p class="card-text">${{ product.price }}</p>

<a href="#" class="btn btn-primary">Add to cart</a>

</div>

</div>

</div>

{% endfor %}

</div>

</div>

</div>

{% endblock %}

**29) Шаблон для деталей товару: detail.html**

Створи файл **products/templates/shop/product/detail.html** і встав у нього наступний код з коментарями:

HTML

{% extends "base.html" %}

{% load static %}

{% block title %}

{{ product.name }}

{% endblock %}

{% block content %}

<div class="row">

<div class="col-md-4">

<img src="{% if product.image %}{% static product.image.url %}{% else %}{% static 'img/no\_image.png' %}{% endif %}" class="img-fluid" alt="{{ product.name }}">

</div>

<div class="col-md-8">

<h1>{{ product.name }}</h1>

<h2><a href="{{ product.category.get\_absolute\_url }}">{{ product.category }}</a></h2>

<p class="price">${{ product.price }}</p>

{{ product.description|linebreaks }}

<a href="#" class="btn btn-primary">Add to cart</a>

</div>

</div>

{% endblock %}

**Пояснення коду**

* {% extends "base.html" %}: Цей тег повідомляє Django, що даний шаблон успадковує структуру з файлу base.html.
* {% load static %}: Цей тег дозволяє використовувати статичні файли (CSS, JS, зображення).
* {{ product.image.url }}: Це звернення до шляху зображення, яке зберігається у моделі.
* {% url 'products:product\_list' %}: Це використання іменованого URL-маршруту, що робить посилання гнучкими.

Твій код файлу products/views.py виправлений та синтаксично правильний. Чудова робота!

**Що далі?**

Коли views налаштовані, наступний крок — створення HTML-шаблонів, які будуть відображати дані. Твої views звертаються до шаблонів shop/product/list.html та shop/product/detail.html.

Нам потрібно створити таку структуру папок для шаблонів у додатку products:

products/

├── templates/

│ └── shop/

│ └── product/

│ ├── list.html # Для списку товарів

│ └── detail.html # Для деталей товару

└── ...

**28) Шаблон для списку товарів: list.html**

Створи файл **products/templates/shop/product/list.html** і встав у нього наступний код з коментарями:

HTML

{% extends "base.html" %}

{% load static %}

{% block title %}

{% if current\_category %}

{{ current\_category.name }}

{% else %}

Products

{% endif %}

{% endblock %}

{% block content %}

<h1>

{% if current\_category %}

{{ current\_category.name }}

{% else %}

Products

{% endif %}

</h1>

<div class="row">

<div class="col-md-3">

<h2>Categories</h2>

<ul class="list-group">

<a href="{% url 'products:product\_list' %}" class="list-group-item {% if not current\_category %}active{% endif %}">All</a>

{% for category in categories %}

<a href="{{ category.get\_absolute\_url }}" class="list-group-item {% if current\_category == category %}active{% endif %}">

{{ category.name }}

</a>

{% endfor %}

</ul>

</div>

<div class="col-md-9">

<div class="row">

{% for product in products %}

<div class="col-md-4 mb-4">

<div class="card h-100 text-center">

<a href="{{ product.get\_absolute\_url }}">

<img class="card-img-top" src="{% if product.image %}{% static product.image.url %}{% else %}{% static 'img/no\_image.png' %}{% endif %}" alt="{{ product.name }}">

</a>

<div class="card-body">

<h5 class="card-title">{{ product.name }}</h5>

<p class="card-text">${{ product.price }}</p>

<a href="#" class="btn btn-primary">Add to cart</a>

</div>

</div>

</div>

{% endfor %}

</div>

</div>

</div>

{% endblock %}

**29) Шаблон для деталей товару: detail.html**

Створи файл **products/templates/shop/product/detail.html** і встав у нього наступний код з коментарями:

HTML

{% extends "base.html" %}

{% load static %}

{% block title %}

{{ product.name }}

{% endblock %}

{% block content %}

<div class="row">

<div class="col-md-4">

<img src="{% if product.image %}{% static product.image.url %}{% else %}{% static 'img/no\_image.png' %}{% endif %}" class="img-fluid" alt="{{ product.name }}">

</div>

<div class="col-md-8">

<h1>{{ product.name }}</h1>

<h2><a href="{{ product.category.get\_absolute\_url }}">{{ product.category }}</a></h2>

<p class="price">${{ product.price }}</p>

{{ product.description|linebreaks }}

<a href="#" class="btn btn-primary">Add to cart</a>

</div>

</div>

{% endblock %}

**Пояснення коду**

* {% extends "base.html" %}: Цей тег повідомляє Django, що даний шаблон успадковує структуру з файлу base.html.
* {% load static %}: Цей тег дозволяє використовувати статичні файли (CSS, JS, зображення).
* {{ product.image.url }}: Це звернення до шляху зображення, яке зберігається у моделі.
* {% url 'products:product\_list' %}: Це використання іменованого URL-маршруту, що робить посилання гнучкими.

**Наступні кроки**

Тепер, коли всі шаблони готові, тобі потрібно запустити сервер і перевірити, чи все працює.

1. **Запусти сервер**:

Bash

python manage.py runserver

1. **Відкрий браузер**: Перейди за адресою http://127.0.0.1:8000.

Ти повинен побачити список товарів з категоріями на бічній панелі. Натискаючи на категорії, ти зможеш фільтрувати товари. Натискання на товар переведе тебе на сторінку з його деталями.

Нам потрібно повідомити Django, де знаходяться твої шаблони. Ми зробимо це, додавши шлях до папки templates у файл налаштувань.

1. Відкрий файл **shop\_SBS\_12\_08\_25/settings.py**.
2. Знайди розділ TEMPLATES (він знаходиться ближче до верху файлу).
3. У списку DIRS додай наступний рядок: os.path.join(BASE\_DIR, 'templates').

Ось як має виглядати виправлений код у settings.py з коментарями:

Python

# settings.py

import os # Цей рядок уже має бути на початку файлу

...

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

# Додаємо шлях до папки з шаблонами

'DIRS': [os.path.join(BASE\_DIR, 'templates')],

'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',

],

},

},

]

...

**Пояснення**

* **os.path.join(BASE\_DIR, 'templates')**: Цей рядок створює повний шлях до папки templates у корені твого проєкту, де знаходиться файл base.html. Тепер Django знатиме, де шукати цей файл.

Після збереження файлу settings.py Django автоматично перезапустить сервер. Перейди за адресою http://127.0.0.1:8000 у браузері, і тепер ти повинен побачити список товарів.

list.html та detail.html також використовують статичні файли (зображення), а ми ще не налаштували, де Django має їх шукати.

**Причина**

Django шукає статичні файли в папках static всередині кожного додатку, але наші зображення знаходяться в products/static/products/images/.

**Виправлення**

Нам потрібно повідомити Django, де шукати статичні файли, додавши шлях до папки static у налаштування.

1. Відкрий файл **shop\_SBS\_12\_08\_25/settings.py**.
2. Знайди розділ STATIC\_URL і додай під ним нові налаштування STATICFILES\_DIRS.

Python

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/5.0/howto/static-files/

STATIC\_URL = 'static/'

# Додаємо шлях до папки 'static' у корені проєкту

STATICFILES\_DIRS = [

os.path.join(BASE\_DIR, 'static'),

]

# Media files (uploaded by user)

MEDIA\_URL = '/media/'

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media/')

1. Збережи файл **settings.py**.

Тепер Django знатиме, де знаходяться твої статичні файли. Спробуй знову оновити сторінку в браузері, і все має запрацювати.

Це означає, що проблема не в налаштуваннях статичних файлів, а, найімовірніше, у шляхах до цих файлів у твоїх шаблонах. Коли ми використовуємо {% static product.image.url %}, Django шукає зображення в папці static, але наш скрипт імпортував шляхи, які вказують на іншу папку: products/images/.

**Причина**

Твій файл products\_data.json містить такі шляхи: products/images/samsung\_s23.webp. Коли ти запускаєш python manage.py collectstatic, ці файли копіюються до папки static, а потім вже звідти підключаються до шаблонів.

Ми цього не зробили.

**Виправлення**

Найпростіший спосіб виправити це — змінити settings.py так, щоб Django шукав файли безпосередньо в кореневій папці додатку products.

1. Відкрий файл **shop\_SBS\_12\_08\_25/settings.py**.
2. Знайди розділ STATICFILES\_DIRS і видали звідти попередній шлях. Додай новий, що вказує на папку static всередині твого додатку products.

Ось як має виглядати виправлений код у settings.py з коментарями:

Python

# settings.py

import os # Цей рядок має бути на початку файлу

...

STATIC\_URL = 'static/'

# Додаємо шлях до папки 'static' всередині додатку 'products'

STATICFILES\_DIRS = [

os.path.join(BASE\_DIR, 'products/static'),

]

...

Після збереження settings.py Django автоматично перезапустить сервер. Онови сторінку в браузері. Тепер Django має знайти твої зображення, і вони повинні відобразитися.

Z

1. Z
2. Z
3. Z
4. Z
5. Z
6. Z
7. Z
8. Z
9. Z
10. Z
11. Z
12. Z
13. Z
14. Z
15. Z
16. Z
17. Z
18. Z
19. Z
20. Z
21. Z
22. Z
23. Z
24. Z
25. Z
26. Z
27. Z
28. Z
29. Z
30. Z
31. Z
32. Z
33. Z
34. Z
35. Z
36. Z
37. Z
38. Z
39. Z
40. Z
41. Z
42. Z
43. Z
44. Z
45. Z
46. Z
47. Z
48. Z
49. Z
50. Z
51. Z
52. Z
53. Z
54. Z
55. Z
56. z