HTTP request/response

REST API

Database cơ bản (MySQL, PostgreSQL, SQLite...)

Python => Django

Tạo dự án đầu tiên thật đơn giản kết nối database mysql, postgreSQL

A screenshot of a black screen

AI-generated content may be incorrect.

Blog cá nhân

Advance:

Class-Based Views (CBV)

Authentication & Authorization

Form nâng cao

Quản lý media & static

Query nâng cao với ORM

Sử dụng Signals

Middleware tùy chỉnh

REST API với Django REST Framework (DRF)

Deploy ứng dụng Django

Testing trong Django

Các chủ đề nâng cao khác (nếu muốn đi sâu)

### 85. Query Performance

**Tóm tắt – Django ORM và Hiệu năng khi Truy vấn**

**1. Query không thực thi ngay**

* Khi bạn viết:

python

CopyEdit

best\_sellers = Book.objects.filter(is\_bestselling=True)

→ Django **chưa truy vấn database ngay lập tức**.

* + Nó chỉ tạo một **QuerySet** – như một bản kế hoạch truy vấn.
  + Chỉ khi bạn **dùng dữ liệu thực sự** (in ra, vòng lặp qua, đếm...), Django mới **gửi truy vấn tới database**.

**2. QuerySet có thể chain**

* Bạn có thể **chuỗi nhiều filter** mà không truy vấn ngay:

python

CopyEdit

amazing\_best\_sellers = best\_sellers.filter(rating\_\_gt=4)

→ Vẫn chưa hit database.

**3. Chỉ truy vấn khi thật sự cần**

* Khi bạn in, duyệt hoặc sử dụng kết quả:

python

CopyEdit

print(best\_sellers)

→ Lúc này Django **mới hit database lần đầu tiên**.

**4. \*\*QuerySet được cache**

* Sau lần đầu truy vấn, kết quả được **cache lại trong bộ nhớ**:
  + In best\_sellers lần 2 → **không hit lại database**
  + amazing\_best\_sellers sử dụng best\_sellers → Django **lọc lại từ kết quả đã cache**

**5. Cẩn thận khi không dùng biến**

* Nếu bạn viết:

python

CopyEdit

print(Book.objects.filter(rating\_\_gt=3))

print(Book.objects.filter(rating\_\_gt=3))

→ Django sẽ **hit database 2 lần** vì bạn **tạo 2 QuerySet khác nhau**, dù cùng nội dung.

* Để tối ưu, bạn nên:

python

CopyEdit

good\_books = Book.objects.filter(rating\_\_gt=3)

print(good\_books)

print(good\_books)

→ Hit database **chỉ 1 lần**.

**✅ Kết luận**

* Django **trì hoãn truy vấn (lazy)** để tăng hiệu năng.
* Biết cách tận dụng cache của QuerySet giúp bạn:
  + Giảm số lần truy vấn
  + Tăng tốc ứng dụng
  + Viết code sạch và tối ưu hơn

### 87. Preparing Templates

**📘 Tóm tắt – Sử dụng Model trong Views và Templates Django**

**1. Tạm biệt interactive shell**

* Shell rất hữu ích để học cách tạo và truy vấn dữ liệu.
* Nhưng thực tế bạn sẽ **dùng model trong views và templates**, không phải shell.

**2. Mục tiêu**

* Hiển thị danh sách sách 📚 trên giao diện.
* Mỗi sách có thể nhấp vào để xem trang chi tiết.
* Tận dụng **template**, **view**, **routing** để xử lý việc này.

**3. Cấu trúc template**

Trong thư mục app (book\_outlet/):

* Tạo templates/book\_outlet/
  + base.html: Template nền (skeleton HTML với {% block %})
  + index.html: Trang hiển thị danh sách sách
  + book\_detail.html: Trang chi tiết sách

Template kế thừa như sau:

django

CopyEdit

{% extends "book\_outlet/base.html" %}

**4. Tạo View trong views.py**

python

CopyEdit

from django.shortcuts import render

def index(request):

return render(request, "book\_outlet/index.html")

**5. Routing: urls.py của app**

Tạo book\_outlet/urls.py:

python

CopyEdit

from django.urls import path

from . import views

urlpatterns = [

path("", views.index, name="index"),

]

**6. Nối vào urls.py của project**

Trong project/urls.py:

python

CopyEdit

from django.urls import path, include

urlpatterns = [

path("", include("book\_outlet.urls")),

]

**7. Chạy server và sửa lỗi đường dẫn**

* Khởi động server: python manage.py runserver
* Nếu gặp lỗi template không tìm thấy → sửa đường dẫn template:  
  "book\_outlet/index.html" thay vì chỉ "index.html"

**8. Tiếp theo**

* Đang hiển thị danh sách tĩnh.
* Mục tiêu tiếp theo: **lấy dữ liệu từ database**, truyền vào template → render danh sách thực tế.

# Note

**Không nên biến mọi field thành index**

* Vì **index giúp tìm nhanh hơn** nhưng sẽ **làm chậm việc ghi dữ liệu**, nên **chỉ đánh index những field hay dùng để truy vấn**, như slug.

**Làm việc với Model trong Django**

* **Model** là cách Django giúp lập trình viên tương tác với cơ sở dữ liệu dễ dàng.
* Chỉ cần định nghĩa một **class kế thừa models.Model**, bạn có thể:
  + Khai báo các **trường dữ liệu** (fields)
  + Chỉ định **kiểu dữ liệu**, **ràng buộc**, **giá trị mặc định**, v.v.
* Dùng **migrations** để Django tạo và đồng bộ bảng dữ liệu với CSDL thật.

**Thao tác với dữ liệu**

Trong **Django shell** hoặc views:

* **Tạo mới**: Book.objects.create(...)
* **Cập nhật**: lấy object và gọi .save()
* **Xóa**: .delete()
* **Lấy dữ liệu**:
  + Tất cả: Book.objects.all()
  + Có điều kiện: .filter(...)
  + Một bản ghi: .get(...)
  + Có xử lý lỗi: get\_object\_or\_404(...)

**Xuất dữ liệu ra giao diện (templates)**

* Dùng vòng lặp {% for book in books %} để hiển thị danh sách.
* Có thể **hiển thị chi tiết một bản ghi**, ví dụ qua view riêng.
* Có thể dùng các phương thức như get\_absolute\_url() để tạo đường dẫn.

**Slug & SEO Friendly URLs**

* Thêm **trường slug** vào model để có URL thân thiện với người dùng & Google.
* Tự động sinh slug bằng cách **override phương thức save()**, dùng slugify(self.title).
* Sử dụng get\_absolute\_url() để sinh URL động dựa trên slug.

**Hiệu năng & Truy vấn**

* Thêm db\_index=True vào những field thường dùng để **truy vấn nhanh hơn**.
* Không nên index tất cả các field vì sẽ **ảnh hưởng hiệu năng khi ghi dữ liệu**.

**Tổng kết**

* Bạn đã học:
  + Định nghĩa model
  + Tạo, cập nhật, xóa, lấy dữ liệu
  + Hiển thị dữ liệu trong template
  + Sử dụng slug cho URL
  + Tối ưu truy vấn bằng index
* Sắp tới sẽ học tiếp phần **Django Admin**, nơi bạn có thể **quản lý dữ liệu qua giao diện web** thay vì shell

# Forms

## Lời dẫn

**Admin Panel** chủ yếu dành cho nhà phát triển hoặc chủ sở hữu website, không phải cho tất cả người dùng.

cần cung cấp cách để người dùng (khách truy cập) nhập dữ liệu thông qua **Forms** và lưu trữ dữ liệu đó (vào database hoặc nơi khác)

## Mục tiêu

* Tạo và xử lý **Forms** để người dùng nhập dữ liệu bên ngoài Admin Panel.
* Tìm hiểu các cách xử lý Forms và đơn giản hóa việc quản lý Forms trong Django.
* Đảm bảo Forms được hiển thị và hoạt động tốt cho khách truy cập website.

## Định nghĩa

Form:

* Tạo biểu mẫu để thu thập dữ liệu người dùng.
* Xử lý dữ liệu đầu vào (input validation, lưu trữ, v.v.).
* Tùy chỉnh và tối ưu hóa Forms cho trải nghiệm người dùng.

Class-Based Views:

* Giới thiệu một cách khác để tạo **Views** (so với Function-Based Views đã học).
* CBVs sẽ hữu ích khi làm việc với Forms và xử lý dữ liệu người dùng.
* CBVs cung cấp cách tổ chức code rõ ràng và tái sử dụng tốt hơn.

## Project feedback app

1. Tạo dự án và ứng dụng mới

django-admin startproject feedback

cd feedback

python manage.py startapp reviews

Mục đích của ứng dụng **Reviews**: Thu thập đánh giá (reviews) từ người dùng về sản phẩm (dự án giả lập, không có sản phẩm thực tế).

1. **Mục tiêu**:

Cho phép người dùng nhập dữ liệu (đánh giá) trực tiếp trên website thông qua **Form**.

Không sử dụng **Admin Panel** cho việc nhập dữ liệu này, vì Admin Panel chỉ dành cho quản trị viên, không phải người dùng thông thường.

1. Quá trình thiết lập ứng dụng

* Tạo thư mục templates/reviews/ trong ứng dụng Reviews => Tạo file template review.html
* Tạo file urls.py trong Reviews app:
  + Thêm url pattern để liên kết với view
* Kết nối url của ứng dụng với dự án
* Thiết lập view và template

 Trong review.html, Form HTML cơ bản đã được tạo với một trường <input> (name="username") và nút <button> để gửi Form.

 Khi nhấn nút "Send", Form được gửi, dữ liệu xuất hiện trong URL dưới dạng query parameters (ví dụ: ?username=Max).

VÌ:

Form sử dụng phương thức **GET** mặc định.

Nút <button> mặc định có type="submit", nên nó tự động gửi Form khi được nhấn.

Nếu đổi type="button" cho nút, Form sẽ không được gửi, và URL không thay đổi.

# Tạo API

pip install djangorestframework

A screenshot of a computer program

AI-generated content may be incorrect.

Thêm serializers.py vào app:

* Biến dữ liệu từ Model → JSON (xuất ra) hoặc JSON → Model (lưu vào DB).

Trong views của app:

Dùng decorator @api\_view để định nghĩa loại request => cho DRF biết view này hỗ trợ phương thức nào

A black screen with white text

AI-generated content may be incorrect.

Cách biết API nên dùng GET, POST, PUT, PATCH hay DELETE?

A screenshot of a phone

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.