1. Model MVT with Django
   1. Tổng quan về MVT trong Django
      1. Django sử dụng mô hình kiến trúc nào? MVT là gì? Nó khác gì với MVC?
      2. Vai trò của từng thành phần trong MVT: Model, View, Template là gì?
      3. Luồng hoạt động của MVT trong Django diễn ra như thế nào?
   2. Model trong Django
      1. Model trong Django là gì? Nó có vai trò gì trong ứng dụng web?
      2. Làm thế nào để định nghĩa một Model trong Django?
      3. Django hỗ trợ những kiểu dữ liệu nào trong Model?
      4. Cách sử dụng makemigrations và migrate để tạo bảng trong database từ Model?
      5. Làm thế nào để thêm, sửa, xóa và truy vấn dữ liệu trong Model?
   3. View trong Django
      1. View trong Django là gì? Nó có nhiệm vụ gì trong mô hình MVT?
      2. Có những cách nào để viết View trong Django?
      3. Cách sử dụng HttpResponse và render template trong View?
      4. Django hỗ trợ truyền dữ liệu từ View sang Template như thế nào?
      5. Cách xử lý request GET và POST trong View?
   4. Template trong Django
      1. Template trong Django là gì? Nó hoạt động như thế nào?
      2. Cú pháp của Django Template Language (DTL) gồm những gì?
      3. Cách truyền biến từ View sang Template như thế nào?
      4. Cách sử dụng Template Inheritance để tái sử dụng giao diện?
   5. Kết nối các thành phần trong MVT
      1. Làm thế nào để kết nối Model, View và Template với nhau?
      2. Cách thiết lập URL trong Django để gọi View?
      3. Django xử lý request từ người dùng và trả về response như thế nào?

--------------------------------------------------------------Trả lời--------------------------------------------------------------------

**Tổng quan về MVT trong Django**

**1. Django sử dụng mô hình kiến trúc nào? MVT là gì? Nó khác gì với MVC?**

✅ **Django sử dụng mô hình kiến trúc MVT (Model - View - Template)**, một biến thể của mô hình phổ biến **MVC (Model - View - Controller)** trong lập trình web.

📌 **Sự khác biệt giữa MVT và MVC:**

| **Kiến trúc** | **Thành phần 1** | **Thành phần 2** | **Thành phần 3** |
| --- | --- | --- | --- |
| **MVC** | Model (Dữ liệu) | View (Giao diện) | Controller (Xử lý logic) |
| **MVT (Django)** | Model (Dữ liệu) | View (Xử lý logic) | Template (Giao diện) |

📍 **Điểm khác biệt chính:**

* Trong **MVC**, **Controller** chịu trách nhiệm xử lý logic và điều hướng dữ liệu.
* Trong **MVT của Django**, **Django framework tự động đóng vai trò Controller**, giúp giảm công việc lập trình viên.

**2. Vai trò của từng thành phần trong MVT: Model, View, Template là gì?**

✔ **Model (M - Dữ liệu & Database)**

* Định nghĩa cấu trúc dữ liệu, tương ứng với bảng trong database.
* Được viết bằng Python thay vì SQL.
* Sử dụng **Django ORM (Object-Relational Mapping)** để làm việc với database.

✔ **View (V - Xử lý logic)**

* Nhận request từ người dùng, xử lý logic nghiệp vụ.
* Giao tiếp với Model để lấy dữ liệu.
* Trả kết quả về Template để hiển thị.

✔ **Template (T - Giao diện người dùng)**

* Chứa mã HTML + Django Template Language (DTL).
* Nhận dữ liệu từ View và hiển thị lên trình duyệt.

🔹 **Ví dụ trực quan về MVT**:  
👉 **User truy cập trang danh sách sinh viên**

1. **Request từ người dùng** → Django gọi **View**
2. **View lấy dữ liệu từ Model** → Trả dữ liệu về **Template**
3. **Template hiển thị giao diện** → Trả kết quả về trình duyệt

**3. Luồng hoạt động của MVT trong Django diễn ra như thế nào?**

💡 **Quy trình xử lý request trong Django theo MVT:**

1️⃣ **Người dùng gửi yêu cầu HTTP** (ví dụ: truy cập /students/).  
2️⃣ **Django nhận request**, tìm URL phù hợp trong urls.py.  
3️⃣ **View được gọi**, xử lý logic và truy vấn dữ liệu từ Model.  
4️⃣ **View gửi dữ liệu tới Template**, render ra trang HTML.  
5️⃣ **Response được trả về cho trình duyệt**, hiển thị nội dung.

**Model trong Django**

**1. Model trong Django là gì? Nó có vai trò gì trong ứng dụng web?**

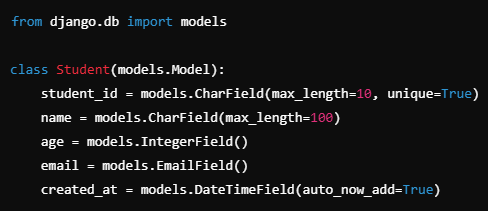
Model trong Django là một lớp Python đại diện cho cấu trúc dữ liệu của ứng dụng. Mỗi model tương ứng với một bảng trong database.

Vai trò của Model trong ứng dụng web:

* Định nghĩa các trường dữ liệu và kiểu dữ liệu.
* Tạo bảng trong database thông qua Django ORM.
* Cung cấp phương thức truy vấn dữ liệu mà không cần viết SQL thuần.
* Đảm bảo tính nhất quán dữ liệu giữa ứng dụng và database.

**2. Làm thế nào để định nghĩa một Model trong Django?**

Một model trong Django là một lớp kế thừa từ django.db.models.Model.



* Mỗi thuộc tính trong class là một cột trong bảng database.
* Các tham số như max\_length, unique, auto\_now\_add giúp kiểm soát dữ liệu.

**3. Django hỗ trợ những kiểu dữ liệu nào trong Model?**

Django cung cấp nhiều kiểu dữ liệu, phổ biến gồm:

* CharField(max\_length=N): Lưu chuỗi văn bản ngắn.
* TextField(): Lưu văn bản dài.
* IntegerField(): Lưu số nguyên.
* FloatField(): Lưu số thực.
* BooleanField(): Lưu giá trị True/False.
* DateTimeField(auto\_now\_add=True): Lưu ngày giờ tạo bản ghi.
* ForeignKey(Model, on\_delete=models.CASCADE): Khóa ngoại liên kết với model khác.

**4. Cách sử dụng makemigrations và migrate để tạo bảng trong database từ Model?**

Sau khi định nghĩa model, cần chạy các lệnh sau để tạo bảng trong database:

1. Tạo migration file từ model:

python manage.py makemigrations

1. Áp dụng migration để tạo bảng trong database:

python manage.py migrate

1. Kiểm tra SQL tương ứng với model:

python manage.py sqlmigrate app\_name migration\_number

**5. Làm thế nào để thêm, sửa, xóa và truy vấn dữ liệu trong Model?**

Django sử dụng ORM để thao tác với dữ liệu.

* **Thêm dữ liệu**:

student = Student(student\_id="SV001", name="Nguyen Van A", age=20, email="a@example.com")

student.save()

* **Lấy danh sách sinh viên**:

students = Student.objects.all()

* **Tìm kiếm theo điều kiện**:

student = Student.objects.get(student\_id="SV001")

* **Cập nhật dữ liệu**:

student.name = "Nguyen Van B"

student.save()

* **Xóa dữ liệu**:

student.delete()

**View trong Django**

**1. View trong Django là gì? Nó có nhiệm vụ gì trong mô hình MVT?**

View trong Django là một hàm hoặc lớp chịu trách nhiệm xử lý logic ứng dụng và trả về kết quả cho người dùng.

Nhiệm vụ của View trong mô hình MVT:

* Nhận yêu cầu từ client (trình duyệt, API).
* Tương tác với Model để lấy dữ liệu từ database.
* Xử lý logic ứng dụng (ví dụ: xác thực người dùng, tính toán dữ liệu).
* Trả về phản hồi, có thể là HTML hoặc JSON thông qua Template hoặc API.

**2. Có những cách nào để viết View trong Django?**

Django hỗ trợ hai cách để viết View:

1. **View dưới dạng hàm (Function-based View - FBV)**
   * Mỗi view là một hàm Python nhận request và trả response.

from django.http import HttpResponse

def home(request):

return HttpResponse("Hello, Django!")

1. **View dưới dạng lớp (Class-based View - CBV)**
   * Sử dụng các lớp được Django cung cấp để tổ chức code tốt hơn.

from django.views import View

from django.http import HttpResponse

class HomeView(View):

def get(self, request):

return HttpResponse("Hello, Django with Class-based View!")

**3. Cách sử dụng HttpResponse và render template trong View?**

* **Sử dụng HttpResponse để trả về nội dung đơn giản:**

from django.http import HttpResponse

def hello(request):

return HttpResponse("Hello, World!")

* **Sử dụng render để hiển thị template:**

from django.shortcuts import render

def home(request):

return render(request, 'home.html')

* + render() tự động tìm file home.html trong thư mục templates/.

**4. Django hỗ trợ truyền dữ liệu từ View sang Template như thế nào?**

Dữ liệu có thể được truyền từ View sang Template thông qua context dictionary.

def home(request):

context = {"name": "Nguyen Van A", "age": 20}

return render(request, 'home.html', context)

Trong template home.html, sử dụng biến như sau:

<p>Name: {{ name }}</p>

<p>Age: {{ age }}</p>

**5. Cách xử lý request GET và POST trong View?**

* **Xử lý request GET:**

def get\_example(request):

param = request.GET.get('name', 'Guest')

return HttpResponse(f"Hello, {param}!")

Truy cập: http://localhost:8000/get\_example?name=John

* **Xử lý request POST:**

def post\_example(request):

if request.method == "POST":

name = request.POST.get('name')

return HttpResponse(f"Received name: {name}")

return HttpResponse("Only POST method allowed")

**Template trong Django**

**1. Template trong Django là gì? Nó hoạt động như thế nào?**

* Template trong Django là các file HTML có thể chứa cú pháp đặc biệt để hiển thị dữ liệu động.
* Template giúp tách biệt phần giao diện với logic xử lý trong View.
* Django sử dụng **Django Template Language (DTL)** để hiển thị dữ liệu và điều khiển logic hiển thị trong template.

**Cách hoạt động:**

1. View xử lý dữ liệu và truyền vào template.
2. Template nhận dữ liệu và hiển thị nội dung phù hợp.
3. Trình duyệt hiển thị kết quả HTML cho người dùng.

Ví dụ:

from django.shortcuts import render

def home(request):

context = {"name": "Nguyen Van A"}

return render(request, 'home.html', context)

Trong home.html:

<p>Xin chào, {{ name }}!</p>

**2. Cú pháp của Django Template Language (DTL) gồm những gì?**

DTL cung cấp một số cú pháp đặc biệt như:

* **Biến ({{ variable }})**: Hiển thị giá trị của biến truyền từ View.
* **Cấu trúc điều kiện ({% if %} ... {% endif %})**

{% if age >= 18 %}

<p>Bạn đủ tuổi.</p>

{% else %}

<p>Bạn chưa đủ tuổi.</p>

{% endif %}

* **Vòng lặp ({% for item in list %} ... {% endfor %})**

<ul>

{% for student in students %}

<li>{{ student.name }}</li>

{% endfor %}

</ul>

* **Kế thừa template ({% extends 'base.html' %})**
* **Chèn nội dung vào block ({% block content %} ... {% endblock %})**

**3. Cách truyền biến từ View sang Template như thế nào?**

* Trong View, sử dụng context để truyền biến vào template:

def home(request):

context = {"title": "Trang Chủ", "user": "Nguyen Van A"}

return render(request, 'home.html', context)

* Trong home.html, sử dụng biến như sau:

<h1>{{ title }}</h1>

<p>Xin chào, {{ user }}!</p>

**4. Cách sử dụng Template Inheritance để tái sử dụng giao diện?**

Template Inheritance giúp tạo bố cục chung và kế thừa lại ở các trang con.

1. **Tạo file base.html chứa giao diện chung:**

<!DOCTYPE html>

<html>

<head>

<title>{% block title %}Trang web{% endblock %}</title>

</head>

<body>

<header>Đây là header</header>

<main>

{% block content %}{% endblock %}

</main>

<footer>Đây là footer</footer>

</body>

</html>

1. **Tạo file home.html kế thừa từ base.html:**

{% extends 'base.html' %}

{% block title %}Trang Chủ{% endblock %}

{% block content %}

<h1>Xin chào, đây là trang chủ!</h1>

{% endblock %}

Khi load home.html, nó sẽ kế thừa base.html và chỉ thay đổi nội dung trong {% block content %}.

**Kết nối các thành phần trong MVT**

**1. Làm thế nào để kết nối Model, View và Template với nhau?**

Trong Django, Model, View và Template kết nối với nhau thông qua các bước sau:

1. **Model**: Đại diện cho dữ liệu trong database.
2. **View**: Lấy dữ liệu từ Model, xử lý logic và gửi đến Template.
3. **Template**: Hiển thị dữ liệu cho người dùng.

Ví dụ:

**1. Định nghĩa Model (models.py)**

from django.db import models

class Student(models.Model):

name = models.CharField(max\_length=100)

age = models.IntegerField()

**2. Tạo View (views.py)**

from django.shortcuts import render

from .models import Student

def student\_list(request):

students = Student.objects.all() # Lấy tất cả sinh viên từ database

return render(request, 'students.html', {'students': students})

**3. Tạo Template (students.html)**

<h2>Danh sách sinh viên</h2>

<ul>

{% for student in students %}

<li>{{ student.name }} - {{ student.age }} tuổi</li>

{% endfor %}

</ul>

**4. Thiết lập URL để gọi View (urls.py)**

from django.urls import path

from .views import student\_list

urlpatterns = [

path('students/', student\_list, name='student\_list'),

]

**2. Cách thiết lập URL trong Django để gọi View?**

Django sử dụng urls.py để định tuyến URL đến View phù hợp.

1. Mở urls.py trong ứng dụng Django.
2. Định nghĩa URL và ánh xạ nó đến một View.
3. Sử dụng path() để khai báo đường dẫn.

Ví dụ:

from django.urls import path

from .views import home

urlpatterns = [

path('', home, name='home'), # Khi truy cập trang chủ, gọi hàm home

]

Khi người dùng truy cập http://localhost:8000/, Django sẽ gọi home View.

**3. Django xử lý request từ người dùng và trả về response như thế nào?**

Django xử lý request theo quy trình sau:

1. **Người dùng gửi request** từ trình duyệt (ví dụ: truy cập http://localhost:8000/students/).
2. **Django kiểm tra urls.py** để tìm URL phù hợp.
3. **Gọi View tương ứng**, xử lý logic và lấy dữ liệu từ Model.
4. **View gửi dữ liệu đến Template** để hiển thị giao diện.
5. **Django trả về response** cho người dùng dưới dạng HTML, JSON hoặc dữ liệu khác.

**REST API trong Django**

**a. Tổng quan về REST API**

i. REST API là gì?  
ii. Tại sao chúng ta cần sử dụng REST API? REST API có lợi ích gì?  
iii. REST khác gì so với các mô hình giao tiếp khác (SOAP, GraphQL,...)?  
iv. REST API có những thành phần chính nào?  
v. HTTP Methods (GET, POST, PUT, DELETE) trong REST API hoạt động như thế nào?

**b. Xây dựng REST API trong Django**

i. Những bước cần thực hiện để xây dựng REST API trong Django?  
ii. Django có hỗ trợ REST API mặc định không? Nếu không thì cần sử dụng gì?  
iii. Django REST Framework (DRF) là gì? Nó hỗ trợ gì cho việc xây dựng API?  
iv. Cách tạo một API đơn giản với Django REST Framework?  
v. Làm thế nào để kiểm tra API sau khi đã xây dựng?

**c. Serializer trong Django REST Framework**

i. Serializer là gì? Nó có vai trò gì trong REST API?  
ii. Tại sao chúng ta cần sử dụng Serializer thay vì trả về dữ liệu trực tiếp?  
iii. Có những loại Serializer nào trong Django REST Framework?  
iv. Làm thế nào để chuyển đổi Model thành JSON bằng Serializer?  
v. Cách sử dụng ModelSerializer để tự động ánh xạ Model với API?

**d. Xử lý Request và Response trong REST API**

i. Làm thế nào để xử lý request GET, POST, PUT, DELETE trong Django REST Framework?  
ii. Cách tạo APIView và ViewSet khác nhau như thế nào? Khi nào nên dùng?  
iii. Làm thế nào để trả về JSON response trong Django REST Framework?  
iv. Cách xử lý lỗi và exception trong REST API?  
v. Làm sao để phân trang dữ liệu khi trả về API?

**e. Xác thực và Phân quyền trong REST API**

i. Có những phương pháp xác thực nào trong Django REST Framework?  
ii. Token Authentication là gì? Làm thế nào để sử dụng nó?  
iii. JWT (JSON Web Token) là gì? Nó khác gì với Token Authentication?  
iv. Làm thế nào để kiểm soát quyền truy cập API bằng Permissions?  
v. Cách bảo vệ API khỏi truy cập trái phép và tấn công bảo mật?

**f. Tối ưu hóa và Triển khai REST API**

i. Làm thế nào để tối ưu REST API để có hiệu suất cao hơn?  
ii. Caching trong Django REST Framework hoạt động như thế nào?  
iii. CORS (Cross-Origin Resource Sharing) là gì? Làm sao để cấu hình CORS trong Django REST Framework?  
iv. Làm thế nào để log và debug REST API hiệu quả?  
v. Các bước triển khai REST API Django lên server thực tế như thế nào?

------------------------------------------------------------Trả lời----------------------------------------------------------------------

**a. Tổng quan về REST API**

**i. REST API là gì?**

REST (Representational State Transfer) API là một kiến trúc giúp các hệ thống giao tiếp với nhau thông qua giao thức HTTP. REST API sử dụng các phương thức HTTP như **GET, POST, PUT, DELETE** để truy xuất và thao tác dữ liệu dưới định dạng **JSON hoặc XML**.

**ii. Tại sao chúng ta cần sử dụng REST API? REST API có lợi ích gì?**

**Lý do sử dụng REST API:**

* Giúp các hệ thống giao tiếp với nhau một cách linh hoạt (ví dụ: frontend Angular có thể giao tiếp với backend Django).
* Không phụ thuộc vào nền tảng, có thể sử dụng với bất kỳ ngôn ngữ lập trình nào.
* Dễ mở rộng và bảo trì nhờ nguyên tắc phân tách rõ ràng giữa client và server.

**Lợi ích:**

* **Đơn giản và nhẹ**: Chỉ sử dụng giao thức HTTP và dữ liệu JSON/XML.
* **Hiệu suất cao**: Không cần giữ kết nối liên tục như SOAP.
* **Khả năng mở rộng tốt**: Có thể dễ dàng tích hợp với các ứng dụng khác.

**iii. REST khác gì so với các mô hình giao tiếp khác (SOAP, GraphQL,...)?**

| **Tiêu chí** | **REST** | **SOAP** | **GraphQL** |
| --- | --- | --- | --- |
| **Kiến trúc** | Dựa trên tài nguyên (Resource) | Dựa trên giao thức RPC/XML | Dựa trên truy vấn dữ liệu (Query) |
| **Giao thức** | HTTP | HTTP, SMTP, TCP | HTTP |
| **Định dạng dữ liệu** | JSON, XML | XML | JSON |
| **Hiệu suất** | Nhanh, nhẹ | Chậm hơn do sử dụng XML | Tối ưu vì chỉ lấy đúng dữ liệu cần |
| **Tính linh hoạt** | Cao, có thể cache dữ liệu | Ít linh hoạt, phải dùng XML | Cao, có thể lấy dữ liệu từ nhiều nguồn trong 1 request |
| **Bảo mật** | OAuth, JWT, HTTPS | WS-Security (tốt hơn nhưng phức tạp) | OAuth, JWT |

REST phổ biến hơn SOAP vì đơn giản, dễ sử dụng và hiệu suất tốt. GraphQL mạnh hơn REST ở khả năng truy vấn dữ liệu, nhưng phức tạp hơn.

**iv. REST API có những thành phần chính nào?**

1. **Client**: Hệ thống hoặc ứng dụng gửi request đến API (ví dụ: trình duyệt web, ứng dụng mobile).
2. **Server**: Hệ thống xử lý request và trả về response (ví dụ: backend Django).
3. **Resource**: Các đối tượng dữ liệu được truy cập thông qua API (ví dụ: user, product, order).
4. **Endpoint**: URL đại diện cho một tài nguyên cụ thể (ví dụ: /api/users/1/).
5. **HTTP Methods**: Các phương thức để thao tác dữ liệu (GET, POST, PUT, DELETE).
6. **Headers**: Thông tin đi kèm với request/response (ví dụ: Authorization, Content-Type).
7. **Status Codes**: Mã phản hồi của server (ví dụ: 200 OK, 404 Not Found).

**v. HTTP Methods (GET, POST, PUT, DELETE) trong REST API hoạt động như thế nào?**

| **HTTP Method** | **Chức năng** | **Ví dụ API** |
| --- | --- | --- |
| **GET** | Lấy dữ liệu từ server | GET /api/users/ (Lấy danh sách users) |
| **POST** | Tạo mới dữ liệu | POST /api/users/ (Thêm user mới) |
| **PUT** | Cập nhật toàn bộ dữ liệu | PUT /api/users/1/ (Cập nhật toàn bộ thông tin user có ID=1) |
| **PATCH** | Cập nhật một phần dữ liệu | PATCH /api/users/1/ (Chỉ cập nhật một số trường của user) |
| **DELETE** | Xóa dữ liệu | DELETE /api/users/1/ (Xóa user có ID=1) |

Mỗi HTTP method giúp API hoạt động như một hệ thống CRUD (**Create, Read, Update, Delete**).

**b. Xây dựng REST API trong Django**

**i. Những bước cần thực hiện để xây dựng REST API trong Django?**

Để xây dựng REST API trong Django, ta thực hiện theo các bước sau:

1. **Cài đặt Django và Django REST Framework (DRF)**:

pip install django djangorestframework

1. **Tạo một dự án Django và ứng dụng mới**:

django-admin startproject myproject

cd myproject

python manage.py startapp myapp

1. **Cấu hình settings.py để thêm rest\_framework vào INSTALLED\_APPS**:

INSTALLED\_APPS = [

...

'rest\_framework',

'myapp',

]

1. **Tạo Model đại diện cho dữ liệu trong cơ sở dữ liệu**.
2. **Tạo Serializer để chuyển đổi dữ liệu Model thành JSON**.
3. **Tạo View và API endpoint để xử lý request**.
4. **Cấu hình URL để API có thể được truy cập**.
5. **Chạy server và kiểm tra API bằng Postman hoặc trình duyệt**.

**ii. Django có hỗ trợ REST API mặc định không? Nếu không thì cần sử dụng gì?**

Django **không hỗ trợ REST API mặc định**, vì nó chủ yếu được thiết kế để xử lý request/response thông qua HTML templates.

Để xây dựng REST API, ta cần sử dụng **Django REST Framework (DRF)**, một thư viện mở rộng giúp tạo API dễ dàng hơn.

**iii. Django REST Framework (DRF) là gì? Nó hỗ trợ gì cho việc xây dựng API?**

**Django REST Framework (DRF)** là một thư viện mạnh mẽ giúp xây dựng REST API trong Django.

**Những tính năng chính của DRF:**

* **Dễ dàng tạo API với serializers và viewsets**.
* **Tích hợp authentication** (JWT, OAuth, Basic Auth,...).
* **Hỗ trợ pagination, filtering, permissions,...**.
* **Tạo API browsable giúp kiểm tra trực tiếp trên trình duyệt**.

**iv. Cách tạo một API đơn giản với Django REST Framework?**

Ví dụ tạo API CRUD cho User.

1. **Định nghĩa Model trong models.py**:

from django.db import models

class User(models.Model):

name = models.CharField(max\_length=100)

email = models.EmailField(unique=True)

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

1. **Tạo Serializer trong serializers.py**:

from rest\_framework import serializers

from .models import User

class UserSerializer(serializers.ModelSerializer):

class Meta:

model = User

fields = '\_\_all\_\_'

1. **Tạo View trong views.py**:

from rest\_framework import viewsets

from .models import User

from .serializers import UserSerializer

class UserViewSet(viewsets.ModelViewSet):

queryset = User.objects.all()

serializer\_class = UserSerializer

1. **Cấu hình URL trong urls.py**:

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

from .views import UserViewSet

router = DefaultRouter()

router.register(r'users', UserViewSet)

urlpatterns = [

path('api/', include(router.urls)),

]

1. **Chạy server và kiểm tra API**:

python manage.py runserver

Truy cập http://127.0.0.1:8000/api/users/ để kiểm tra API.

**v. Làm thế nào để kiểm tra API sau khi đã xây dựng?**

Có nhiều cách để kiểm tra API:

1. **Sử dụng trình duyệt**: DRF cung cấp giao diện API browsable.
2. **Sử dụng Postman hoặc cURL** để gửi request.
   * **Postman**: Gửi request GET, POST, PUT, DELETE để kiểm tra API.
   * **cURL**:

curl -X GET http://127.0.0.1:8000/api/users/

1. **Viết unit test trong Django** để kiểm tra API tự động.

**c. Serializer trong Django REST Framework**

**i. Serializer là gì? Nó có vai trò gì trong REST API?**

Serializer trong Django REST Framework (DRF) có chức năng **chuyển đổi dữ liệu** giữa các dạng khác nhau, cụ thể là:

* **Từ Model Django → JSON (để gửi dữ liệu ra API)**.
* **Từ JSON → Model Django (để lưu dữ liệu vào database)**.

Vai trò chính của Serializer trong REST API:

* **Giúp API trả về dữ liệu có định dạng JSON** để client (React, Angular, mobile app,...) có thể sử dụng.
* **Hỗ trợ validate dữ liệu khi nhận request từ client**.
* **Cung cấp cơ chế ánh xạ dữ liệu giữa database và API một cách tự động**.

**ii. Tại sao chúng ta cần sử dụng Serializer thay vì trả về dữ liệu trực tiếp?**

Nếu không dùng Serializer, ta phải **tự viết logic** để chuyển đổi Model Django thành JSON, điều này rất tốn công sức.

Ví dụ **không dùng Serializer**:

python

CopyEdit

from django.http import JsonResponse

from .models import User

def user\_list(request):

users = User.objects.all()

data = [{"id": user.id, "name": user.name, "email": user.email} for user in users]

return JsonResponse(data, safe=False)

Bằng cách dùng Serializer, ta có thể viết **code ngắn gọn hơn** và DRF sẽ lo phần chuyển đổi:

python

CopyEdit

from rest\_framework.response import Response

from rest\_framework.decorators import api\_view

from .models import User

from .serializers import UserSerializer

@api\_view(['GET'])

def user\_list(request):

users = User.objects.all()

serializer = UserSerializer(users, many=True)

return Response(serializer.data)

Như vậy, **Serializer giúp tiết kiệm thời gian, đảm bảo tính nhất quán và hỗ trợ validation dữ liệu**.

**iii. Có những loại Serializer nào trong Django REST Framework?**

DRF cung cấp **hai loại Serializer chính**:

1. **Serializer (thủ công)** – Tự định nghĩa các field, phương thức chuyển đổi:

python

CopyEdit

from rest\_framework import serializers

class UserSerializer(serializers.Serializer):

id = serializers.IntegerField()

name = serializers.CharField(max\_length=100)

email = serializers.EmailField()

1. **ModelSerializer (tự động)** – Dựa trên Model Django, tự ánh xạ các field:

python

CopyEdit

from rest\_framework import serializers

from .models import User

class UserSerializer(serializers.ModelSerializer):

class Meta:

model = User

fields = '\_\_all\_\_'

ModelSerializer giúp **giảm code lặp lại**, do nó tự động lấy field từ Model.

**iv. Làm thế nào để chuyển đổi Model thành JSON bằng Serializer?**

Giả sử ta có Model User:

python

CopyEdit

from django.db import models

class User(models.Model):

name = models.CharField(max\_length=100)

email = models.EmailField(unique=True)

Ta tạo Serializer để chuyển Model thành JSON:

python

CopyEdit

from rest\_framework import serializers

from .models import User

class UserSerializer(serializers.ModelSerializer):

class Meta:

model = User

fields = '\_\_all\_\_'

Sử dụng Serializer trong View để trả về JSON:

python

CopyEdit

from rest\_framework.response import Response

from rest\_framework.decorators import api\_view

from .models import User

from .serializers import UserSerializer

@api\_view(['GET'])

def user\_list(request):

users = User.objects.all()

serializer = UserSerializer(users, many=True)

return Response(serializer.data)

Khi truy cập API, dữ liệu sẽ được tự động chuyển thành JSON:

json

CopyEdit

[

{"id": 1, "name": "John Doe", "email": "john@example.com"},

{"id": 2, "name": "Jane Doe", "email": "jane@example.com"}

]

**v. Cách sử dụng ModelSerializer để tự động ánh xạ Model với API?**

Sử dụng **ModelSerializer** giúp giảm code lặp lại bằng cách tự ánh xạ Model Django với API.

1. **Định nghĩa ModelSerializer**:

python

CopyEdit

from rest\_framework import serializers

from .models import User

class UserSerializer(serializers.ModelSerializer):

class Meta:

model = User

fields = '\_\_all\_\_'

1. **Sử dụng ModelSerializer trong ViewSet**:

python

CopyEdit

from rest\_framework import viewsets

from .models import User

from .serializers import UserSerializer

class UserViewSet(viewsets.ModelViewSet):

queryset = User.objects.all()

serializer\_class = UserSerializer

1. **Cấu hình URL để API hoạt động**:

python

CopyEdit

from django.urls import path, include

from rest\_framework.routers import DefaultRouter

from .views import UserViewSet

router = DefaultRouter()

router.register(r'users', UserViewSet)

urlpatterns = [

path('api/', include(router.urls)),

]

1. **Truy cập API để xem kết quả**:
   * GET /api/users/ → Lấy danh sách user
   * POST /api/users/ → Tạo user mới
   * PUT /api/users/1/ → Cập nhật user có ID = 1
   * DELETE /api/users/1/ → Xóa user có ID = 1