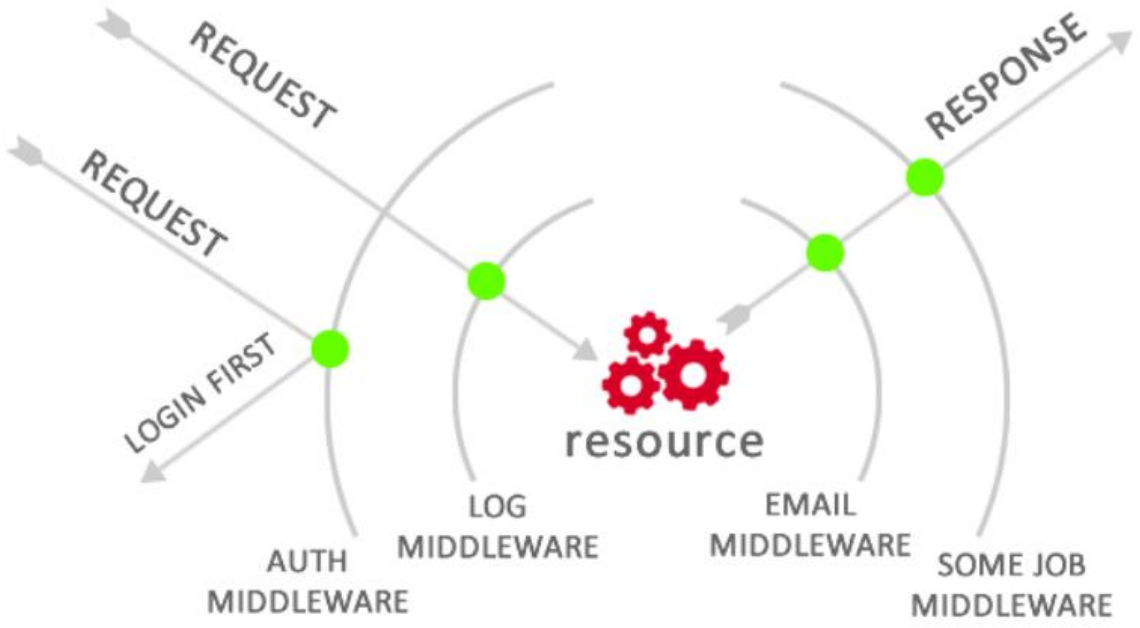
1. Middleware? custom middleware, có bao nhiêu loại? tác dụng?

- Là một đoạn mã logic nằm giữa request và respond. Sau khi nhận request nó sẽ "tiền xử lí" sau đó gửi cho 1 middleware khác hoặc gửi tới logic chính(controller hoặc view trong Django). Sauk hi nhận được respond từ logic chính có thể qua 1 middleware khác, hoặc gửi data cho người dùng.

- được khai báo trong setting.py

- Mỗi custom middleware được viết thành 1 lớp



Tác dụng của middleware:

- Tái sử dụng code áp dụng cho nhiều view khác nhau
- Tách biệt logic phụ với logic chính trong view
- Các hoạt động như cache hoặc kiểm tra quyền truy cập thực hiện tại 1 điểm duy nhất, tăng performance

Phân loại:

Các loại Middleware:

1. **Xử lý Request**:
   - **Xác thực**: Kiểm tra xem người dùng đã đăng nhập chưa.
   - **Ghi log**: Lưu lại các thông tin về request vào cơ sở dữ liệu.
   - **Thêm thông tin**: Chèn thêm các thông tin vào request như thông tin về người dùng.
2. **Xử lý Response**:
   - **Ghi log**: Ghi lại thông tin của response trả về.
   - **Cache**: Thêm hoặc xử lý bộ nhớ đệm cho response.
   - **Chỉnh sửa Header**: Thêm hoặc chỉnh sửa các header HTTP trước khi trả về.

Dưới đây là thứ tự các build-in middleware được suggest:

1. SecurityMiddleware
2. UpdateCacheMiddleware
3. GZipMiddleware
4. SessionMiddleware
5. ConditionalGetMiddleware
6. LocaleMiddleware
7. CommonMiddleware
8. CsrfViewMiddleware
9. AuthenticationMiddleware
10. MessageMiddleware
11. FetchFromCacheMiddleware
12. FlatpageFallbackMiddleware
13. RedirectFallbackMiddleware

Middleware	Dùng cho	Hoạt động với
`SecurityMiddleware`	Request	Tăng cường bảo mật
`SessionMiddleware`	Request	Quản lý session
`CommonMiddleware`	Request	Chức năng chung
`CsrfViewMiddleware`	Request	CSRF Protection
`AuthenticationMiddleware`	Request	Xác thực người dùng
`MessageMiddleware`	Request, Response	Quản lý thông điệp
`XFrameOptionsMiddleware`	Response	Ngăn clickjacking
`CorsMiddleware`	Request, Response	Hỗ trợ CORS
`LocaleMiddleware`	Request	Địa phương hóa
`StaticFilesMiddleware`	Request	Phục vụ tệp tĩnh
`SecurityMiddleware` (sửa đổi)	Request	Chính sách bảo mật

2. Lifecycle request django, nó đi qua những gì?

Request Django -> middleware(kiểm tra authen, ghi log request, redirect nếu không có quyền truy cập, thêm thông tin vào request) -> xử lí logic chính (view) -> middleware(ghi log respond, thêm header vào respond) -> respond

3. Các loại orm sử dụng trong django? Ưu nhược điểm

Django's Object-Relational Mapper (ORM) is a core component of the Django web framework. QuerySets is powerful API to interact with DB

Có 2 loại ORM phổ biến sử dụng trong các dự án Django:

- Active record: Model tương ứng trực tiếp tới 1 bảng trong DB. Các phương thức truy vấn gắn liền trực tiếp với đối tượng đó. **Ưu điểm:** đơn giản, dễ sử dụng. **Nhược điểm:** Sự phụ thuộc giữa đối tượng và DB gây khó khăn truy vấn trong các dự án phức tạp (**Django ORM**)

- Data mapper: tách biệt đối tượng khỏi DB, có một lớp trung gian Data mapper ánh xạ giữa đối tượng và db. **Ưu điểm:** tách biệt sự phụ thuộc giữa đối tượng và DB giúp thiết kế DB linh hoạt hơn, dễ dàng kiểm thử mà không liên quan tới db thật**, Nhược điểm:** phức tạp. (SQLALchemy)

- Benefits:

Tăng tốc độ phát triển vì không cần viết SQL thủ công

Dễ dàng thay đổi giữa các DB khác nhau

Tính năng built in: admin interface, forms, authentication

Giảm lỗi

4. decorator sử dụng trong authorization

5. demo sử dụng session, jwt? so sánh

Session: người dùng nhập user ps, sau đó kiểm tra bằng hàm authenticate() , sau khi xác thực hàm login(user, ps) tạo một session mới duy nhất trên máy chủ, sau đó Django gửi cho người dùng session cookie chứa ID duy nhất được lưu trong browser. Trong lần gửi request tiếp theo browser tự động gửi cookie session này tới máy chủ. Sau đó server sử dụng ID của session này tìm lại phiên tương ứng với người dùng nào.

Đăng ký bằng Session-based (qua form HTML):

- View của bạn nhận dữ liệu từ CustomUserCreationForm.
- Khi gọi form.is_valid() và form.save(), Django sẽ sử dụng dữ liệu hợp lệ đó để gọi hàm User.objects.create_user().
- Hàm này sẽ tạo một đối tượng User mới và tự động **hash (mã hóa) mật khẩu** trước khi lưu vào database.

Đăng ký bằng JWT (qua API):

- View nhận dữ liệu JSON.
- **UserSerializer** đóng vai trò tương tự như CustomUserCreationForm. Nó xác thực dữ liệu.
- Trong phương thức create() của UserSerializer, nó cũng gọi trực tiếp User.objects.create_user().
- Điều này đảm bảo mật khẩu cũng được **hash và lưu trữ an toàn** giống session.

6. Đọc kĩ thư viện unittesing

unittest.TestCase: Lớp cơ sở của thư viện unittest của Python. Nếu test của bạn không cần tương tác với database hoặc các tính năng đặc thù của Django, bạn có thể sử dụng lớp này.

django.test.SimpleTestCase: Lớp cơ sở để viết các test không tương tác với database. Ví dụ: kiểm tra URL resolver, hoặc một số logic đơn giản không cần lưu trữ dữ liệu.

django.test.TestCase: Đây là lớp phổ biến nhất và được khuyến nghị sử dụng cho hầu hết các test trong Django. Nó kế thừa từ SimpleTestCase và cung cấp một test client (self.client) để mô phỏng request HTTP. **Quan trọng:** Mỗi test method trong TestCase được chạy trong một transaction database riêng biệt và được rollback sau khi hoàn thành, đảm bảo sự cô lập và trạng thái sạch sẽ cho mỗi test.

django.test.TransactionTestCase: Được sử dụng khi bạn cần kiểm thử các hành vi liên quan đến transaction database cụ thể. Khác với TestCase, nó không rollback database sau mỗi test method, mà thay vào đó, nó sẽ TRUNCATE (xóa toàn bộ dữ liệu) các bảng sau mỗi test, điều này chậm hơn nhưng cần thiết cho một số trường hợp test transaction phức tạp.

django.test.LiveServerTestCase: Dùng để kiểm thử các tương tác với trình duyệt web thực sự (ví dụ: với Selenium). Nó khởi động một server phát triển Django trong một thread riêng để bạn có thể gửi yêu cầu HTTP đến nó từ các công cụ kiểm thử bên ngoài.

Chạy Test

- Sử dụng lệnh: python manage.py test
- **Chạy toàn bộ project:** python manage.py test
- **Chạy một ứng dụng cụ thể:** python manage.py test myapp
- **Chạy một test class cụ thể:** python manage.py test myapp.tests.MyTestClass
- **Chạy một phương thức test cụ thể:** python manage.py test myapp.tests.MyTestClass.test_my_method
- **Các tùy chọn dòng lệnh:**
    - --verbosity 2: Hiển thị chi tiết hơn về các test đang chạy.
    - --failfast: Dừng chạy test ngay lập tức khi phát hiện lỗi đầu tiên.
    - --keepdb: Giữ lại test database sau khi chạy test (để kiểm tra thủ công hoặc debug).
    - --parallel [N]: Chạy test song song trên N processes (N mặc định là số lượng CPU core).

7. socketIO dựng sever, tạo app chạy 2 client chat với nhau

8. solid

S – Single responsibility Principle

O – Open/closed Principle

L – Liskov substitution Principle

I – Interface segregation Principle

D – Depencency Inversion Principle

1. Single Responsibility Principle

Nguyên tắc này mô tả rằng mỗi class hoặc 1 mô đun chỉ thực hiện 1 chức năng duy nhất, giúp dễ dàng mở rộng và bảo trì mã.

2. Open/Closed Principle

“Open for extension, but closed for modification” – 1 module, class, func được thiết kế để có thể dễ dàng mở rộng thêm những chức năng mới bằng cách kế thừa để khai báo những chức năng mới, mà không thay đổi trực tiếp mã trong module gốc

3. Liskov substitution Principle

Nguyên tắc thay thế Liskov: 1 class con được thiết kế để hoàn toàn có thể thay thế class cha, nếu class con không đạt được tiêu chí này thì nó vi phạm nguyên tắc này. Ví dụ class cha là Duck, các class con như SkyDuck, MuskovyDuck, hoặc ToyDuck khi đó ToyDuck không hoạt động vì có cần pin để hoạt động, và không thể bay.

4. Interface segregation Principle

Một interface cần được chia nhỏ thành các interface khác nhau dựa trên chức năng của các phương thức. Nhằm mục đích khi thực thi interface và định nghĩa các phương thức của interface đều được sử dụng, tránh TH định nghĩa mà không sử dụng.

5. Dependency Inversion Principle

Nguyên lý đảo ngược sự phụ thuộc:
- Các module cấp cao không phụ thuộc vào các module cấp thấp mà chúng nên phụ thuộc vào interface (abstraction)

- Các interface(abstraction) không phụ thuộc vào các chi tiết của nó mà ngược lại

Ví dụ: Có 2 loại bóng đèn là đèn huỳnh quang và đèn dây tóc đều có chung loại đui xoáy. Nguồn điện ở đây là module cấp cao, 2 bóng đèn là 2 module cấp thấp, đui đèn là interface. Module cấp cao giao tiếp với module cấp thấp thông qua interface, nó không quan tâm bóng đèn là loại nào, chỉ quan tâm interface có phù hợp hay không.

9. design pattern (có 3 cái category, code demo mỗi nhóm 3 ví dụ)

10. mongoDB

THÔNG TIN KHÁC

Các lệnh với manage.py

Lệnh	Mô tả
python manage.py runserver	Chạy server phát triển của Django.
python manage.py makemigrations	Tạo các migration cho thay đổi trong models.
python manage.py migrate	Áp dụng migration vào cơ sở dữ liệu.
python manage.py createsuperuser	Tạo người dùng admin để truy cập vào Django admin.
python manage.py shell	Mở shell Python với môi trường Django để thử nghiệm mã Python.
python manage.py startapp <app_name>	Tạo một ứng dụng mới trong dự án Django.
python manage.py startproject <name>	Tạo một dự án Django mới.
python manage.py test	Chạy các bài kiểm tra đã định nghĩa trong tests.py.
python manage.py collectstatic	Thu thập tất cả tệp tĩnh vào thư mục STATIC_ROOT cho môi trường sản xuất.
python manage.py dbshell	Mở shell cơ sở dữ liệu (nếu cấu hình cơ sở dữ liệu hỗ trợ).
python manage.py showmigrations	Hiển thị trạng thái của tất cả các migration trong dự án.
python manage.py flush	Xóa tất cả dữ liệu trong cơ sở dữ liệu và reset các bảng.
python manage.py migrate <app_name>	Áp dụng migration cho một app cụ thể.
python manage.py check	Kiểm tra cấu hình của dự án Django.
python manage.py version	Hiển thị phiên bản của Django hiện tại.

Các kiểu dữ liệu trong model

Kiểu Dữ Liệu	Mô Tả	Ví Dụ
CharField	Lưu trữ chuỗi văn bản ngắn (yêu cầu max_length).	name = models.CharField(max_length=100)
TextField	Lưu trữ chuỗi văn bản dài.	description = models.TextField()
IntegerField	Lưu trữ số nguyên (integer).	age = models.IntegerField()
PositiveIntegerField	Lưu trữ số nguyên dương.	stock_quantity = models.PositiveIntegerField()
FloatField	Lưu trữ số thực (floating point).	weight = models.FloatField()
DecimalField	Lưu trữ số thực với độ chính xác cao, thường dùng cho tiền tệ.	price = models.DecimalField(max_digits=10, decimal_places=2)
DateField	Lưu trữ ngày (yyyy-mm-dd).	birthdate = models.DateField()
DateTimeField	Lưu trữ ngày và giờ (yyyy-mm-dd hh:mm:ss).	created_at = models.DateTimeField(auto_now_add=True)
TimeField	Lưu trữ thời gian (hh:mm:ss).	meeting_time = models.TimeField()
BooleanField	Lưu trữ giá trị boolean (True/False).	is_active = models.BooleanField(default=True)
EmailField	Lưu trữ địa chỉ email, kiểm tra định dạng hợp lệ.	email = models.EmailField()
URLField	Lưu trữ URL (địa chỉ web).	website = models.URLField()
FileField	Lưu trữ tệp (file), yêu cầu chỉ định upload_to.	file = models.FileField(upload_to='uploads/')
ImageField	Lưu trữ tệp hình ảnh, yêu cầu cài đặt thư viện Pillow.	image = models.ImageField(upload_to='images/')
ForeignKey	Lưu trữ khóa ngoại (mối quan hệ một-nhiều).	category = models.ForeignKey(Category, on_delete=models.CASCADE)
ManyToManyField	Lưu trữ mối quan hệ nhiều-nhiều.	tags = models.ManyToManyField(Tag)
OneToOneField	Lưu trữ mối quan hệ một-một giữa các model.	profile = models.OneToOneField(Profile, on_delete=models.CASCADE)
SlugField	Lưu trữ chuỗi slug (dùng trong URL, không dấu).	slug = models.SlugField(unique=True)
UUIDField	Lưu trữ UUID (Unique Universal Identifier).	unique_id = models.UUIDField(default=uuid.uuid4, editable=False)
JSONField	Lưu trữ dữ liệu JSON (dành cho Django 3.1 trở lên).	data = models.JSONField()

Các gói thư viện có sẵn hay dùng trong dự án Django

django.contrib

django.forms

django.db

django.middleware

django.shortcuts

django.urls

django.template

Django.apps

Django.test

Một số gói mở rộng:
django-rest-framework: công cụ xây dựng API Restful

django-allauth: gói đăng ký đăng nhập OAuth, qua mạng xã hội

django-celery: công cụ chạy tác vụ nền: gửi email hoặc xử lý công việc không đồng bộ

django-redis: tích hợp redis lưu trữ session, cache, queue

django-crispy-forms: tạo các forms với giao diện đẹp mắt

Các gói trong django.contrib:

django.contrib.auth

django.contrib.admin

django.contrib.sessions

django.contrib.messages

django.contrib.staticfiles

django.contrib.contenttypes

django.contrib.sites

django.contrib.humanize

django.contrib.redirects

django.contrib.flatpages

Các gói trong django.contrib.auth:

1. **django.contrib.auth.models**

- User: Mô hình người dùng mặc định.
- Group: Mô hình nhóm người dùng, cho phép phân quyền theo nhóm.
- Permission: Mô hình quyền, cho phép phân quyền cho người dùng hoặc nhóm.
- Các phương thức hữu ích: authenticate(), login(), logout(), create_user(), create_superuser(), v.v.

2. **django.contrib.auth.forms:** Cung cấp các form dùng cho việc đăng ký, đăng nhập và thay đổi thông tin người dùng. UserCreationForm: Form dùng để đăng ký người dùng mới.

- AuthenticationForm: Form dùng để đăng nhập người dùng.
- PasswordChangeForm: Form dùng để thay đổi mật khẩu người dùng.
- PasswordResetForm: Form dùng để gửi email đặt lại mật khẩu.
- SetPasswordForm: Form dùng để thiết lập lại mật khẩu cho người dùng.

3**. django.contrib.auth.backends:** Cung cấp các backend xác thực. Các backend này xác định cách xác thực người dùng (ví dụ, xác thực qua cơ sở dữ liệu, qua LDAP, qua OAuth, v.v.).

  - ModelBackend: Sử dụng cơ sở dữ liệu của Django để xác thực người dùng.

  Sử dụng trong dự án thực tế: Dễ dàng mở rộng để hỗ trợ các phương thức xác thực khác ngoài ModelBackend, như xác thực qua mạng xã hội hoặc qua các hệ thống bên ngoài.

4. django.contrib.auth.middleware: Cung cấp các lớp middleware giúp xử lý yêu cầu của người dùng liên quan đến xác thực và phân quyền.

  - AuthenticationMiddleware: Middleware này đảm bảo rằng đối tượng user sẽ có sẵn trong request (là người dùng đã đăng nhập, nếu có).
  - SessionAuthenticationMiddleware: Cung cấp middleware xác thực qua session (dùng trong đăng nhập và duy trì phiên người dùng).

5. django.contrib.auth.signals: Cung cấp các tín hiệu (signals) để xử lý các sự kiện liên quan đến người dùng như đăng nhập, đăng ký, thay đổi mật khẩu, v.v.

  - user_logged_in: Được phát ra khi người dùng đăng nhập thành công.
  - user_logged_out: Được phát ra khi người dùng đăng xuất.
  - user_password_changed: Được phát ra khi người dùng thay đổi mật khẩu.
  - user_login_failed: Được phát ra khi một lần đăng nhập không thành công.
- **Sử dụng trong dự án thực tế**: Bạn có thể sử dụng các tín hiệu này để thực hiện các hành động bổ sung khi người dùng đăng nhập, đăng xuất hoặc thay đổi mật khẩu (ví dụ: ghi lại lịch sử đăng nhập, gửi thông báo, v.v.).

6. django.contrib.auth.views: Cung cấp các view sẵn có để xử lý các tác vụ liên quan đến xác thực người dùng như đăng nhập, đăng xuất, thay đổi mật khẩu, và đặt lại mật khẩu.

  - LoginView: View đăng nhập người dùng.

    - LogoutView: View đăng xuất người dùng.
    - PasswordChangeView: View cho phép người dùng thay đổi mật khẩu.
    - PasswordResetView: View cho phép người dùng yêu cầu đặt lại mật khẩu qua email.
    - PasswordResetConfirmView: View cho phép người dùng đặt lại mật khẩu qua link gửi email.
- **Sử dụng trong dự án thực tế**: Các view này giúp tiết kiệm thời gian khi bạn không cần phải tự tạo view và form cho việc đăng nhập, đăng xuất, hoặc thay đổi mật khẩu người dùng.

7. django.contrib.auth.management: Cung cấp các lệnh quản lý cho Django admin, ví dụ như tạo superuser hoặc kiểm tra và thay đổi mật khẩu người dùng.

    - createsuperuser: Lệnh dùng để tạo tài khoản superuser (quản trị viên).
    - changepassword: Lệnh để thay đổi mật khẩu của người dùng từ command line.

Các phương thức trong Django.contrib.auth.models

- **authenticate()**: Kiểm tra tính hợp lệ của người dùng.
- **login()**: Đăng nhập người dùng vào session.
- **logout()**: Đăng xuất người dùng.
- **is_authenticated**: Kiểm tra xem người dùng đã đăng nhập hay chưa.
- **set_password()**: Mã hóa mật khẩu người dùng.
- **check_password()**: Kiểm tra mật khẩu của người dùng.
- **get_username()**: Trả về tên đăng nhập của người dùng.
- **user_permissions**: Truy vấn các quyền của người dùng.
- **get_full_name()**: Trả về tên đầy đủ của người dùng (first_name + last_name).
- **get_short_name()**: Trả về tên ngắn gọn (first_name).
- **has_perm(permission)**: Kiểm tra quyền của người dùng.
- **has_perms(permissions)**: Kiểm tra xem người dùng có tất cả quyền hay không.
- **has_module_perms(module_name)**: Kiểm tra quyền truy cập vào một module.
- **set_unusable_password()**: Thiết lập mật khẩu không thể sử dụng được.
- **delete()**: Xóa người dùng khỏi cơ sở dữ liệu.
- **is_superuser**: Kiểm tra xem người dùng có phải là quản trị viên không.
- **groups**: Lấy các nhóm mà người dùng thuộc về.
- **user_permissions**: Lấy các quyền mà người dùng có.
-

Phân chia theo từng nhóm sử dụng với từng đối tượng được định nghĩa sẵn trong models:

User:

- **create_user()**: Tạo người dùng mới.
- **create_superuser()**: Tạo superuser (quản trị viên).
- **set_password()**: Mã hóa mật khẩu người dùng.
- **check_password()**: Kiểm tra mật khẩu người dùng.
- **is_authenticated**: Kiểm tra người dùng đã đăng nhập chưa.
- **has_perm(permission)**: Kiểm tra quyền của người dùng.

- **has_perms(permissions)**: Kiểm tra tất cả quyền của người dùng.
- **groups**: Trả về các nhóm mà người dùng thuộc về.

Group:

- **add()**: Thêm người dùng vào nhóm.
- **remove()**: Loại bỏ người dùng khỏi nhóm.
- **clear()**: Xóa tất cả người dùng khỏi nhóm.
- **permissions**: Trả về các quyền của nhóm.
- **has_perm(permission)**: Kiểm tra quyền của nhóm.

Permission:

- **create()**: Tạo quyền mới.
- **get_codename()**: Trả về codename của quyền.
- **assign()**: Liên kết quyền với người dùng hoặc nhóm.

Tóm tắt về các thuộc tính trong Meta trong class models:

db_table: Xác định tên bảng trong cơ sở dữ liệu.

ordering: Đặt thứ tự mặc định cho các đối tượng khi truy vấn.

verbose_name và verbose_name_plural: Cung cấp tên mô hình trong giao diện người dùng (Admin).

unique_together: Đảm bảo sự kết hợp của các trường là duy nhất trong cơ sở dữ liệu.

index_together: Tạo chỉ mục cho sự kết hợp của các trường trong cơ sở dữ liệu.

constraints: Định nghĩa các ràng buộc cơ sở dữ liệu tùy chỉnh.

default_permissions: Xác định các quyền mặc định cho mô hình.

permissions: Định nghĩa quyền tùy chỉnh cho mô hình.

app_label: Xác định app mà mô hình này thuộc về.

Các thuộc tính trong Meta trong class Serializers:

model: Chỉ định mô hình mà serializer sẽ hoạt động trên đó.

fields: Danh sách các trường mà serializer sẽ xử lý.

exclude: Danh sách các trường không muốn bao gồm trong serializer.

read_only_fields: Các trường chỉ có thể đọc, không thể ghi vào.

write_only_fields: Các trường chỉ có thể ghi vào, không thể đọc.

depth: Độ sâu của serializer khi xử lý quan hệ (foreign key, many-to-many).