**1.tích hợp socketIO cùng backend - có thể giao tiếp với backend, lấy dữ liệu từ BD (cách sử dụng messageQ để kết nối giữa server backend và socketIO)**

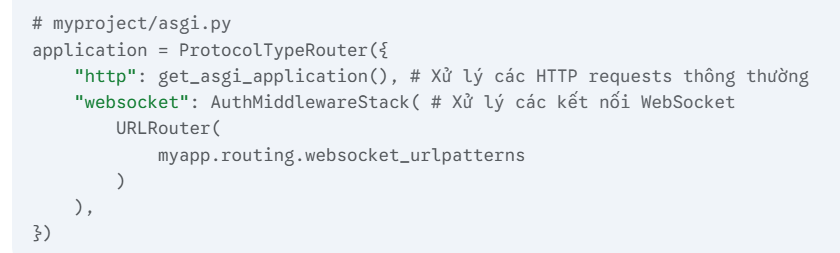
**Wsgi(web server gateway interface - *Gunicorn, uWSGI*):** tiêu chuẩn giao diện **đồng bộ** giữa web server(Nginx) và ứng dụng Python(Django). Mỗi 1 request được xử lí bởi một thread hoặc 1 process riêng biệt trên server

**Asgi (asynchronous server gateway interface *- Daphne, Uvicorn, Hypercorn*)**: tiêu chuẩn giao diện bất đồng bộ, thiết kế xử lí cả ứng dụng đồng bộ/ bất đồng bộ. Xây dựng để hỗ trợ WebSockets, HTTP/2,… các giao thức bất đồng bộ khác.

- WebSocket Handshake: mọi kết nối websocket đều bắt dầu từ HTTP thông thường từ client tới server nhưng request này có header đặc biệt để thông báo với server rằng nó muốn nâng cấp lên giao thức websocket

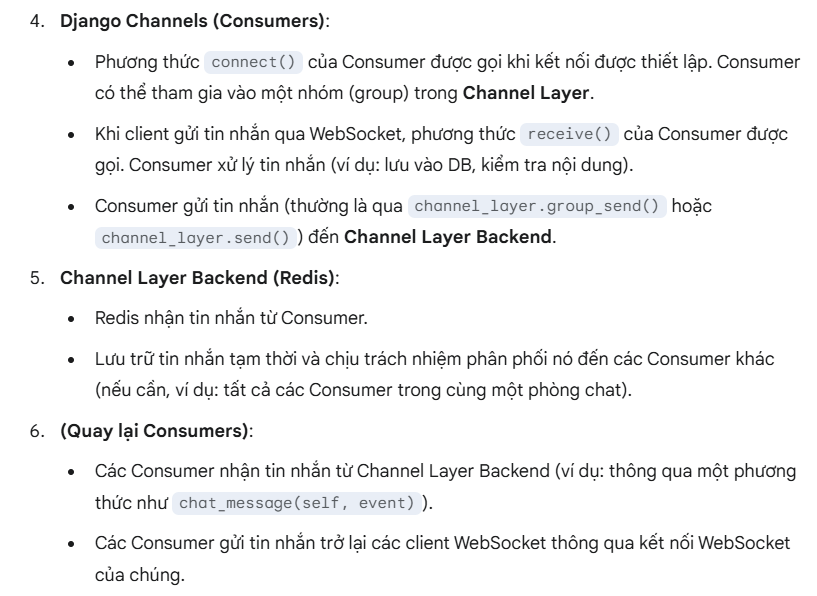
- Sau đó ASGI (daphne) nhận request đó và kiểm tra trong header xem đó có phải là 1 request để thiết lập kết nối websocket hay không

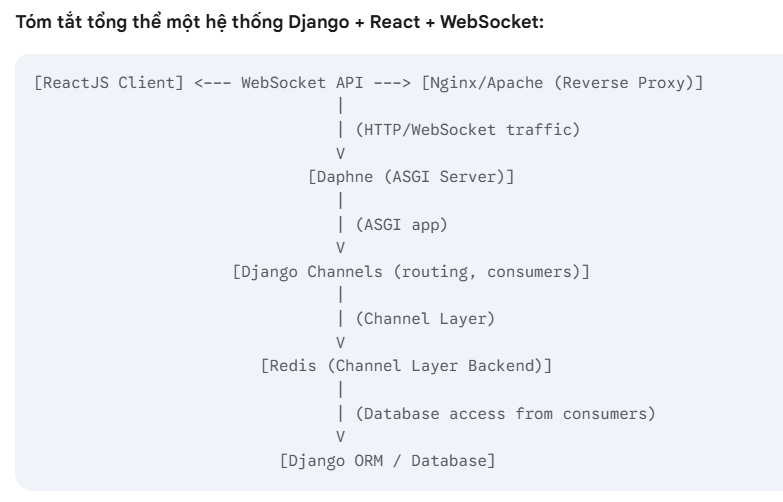
- Nếu đúng là request thiết lập websocket thì server trả về reponse với trạng thái *101 Switching Protocols.* Sau khi client nhận được phản hồi này thì kết nôi TCP không còn được sử dụng cho giao thức HTTP nữa, nó dùng giao thức Websocket để thay thế từ thời điểm này trở đi.

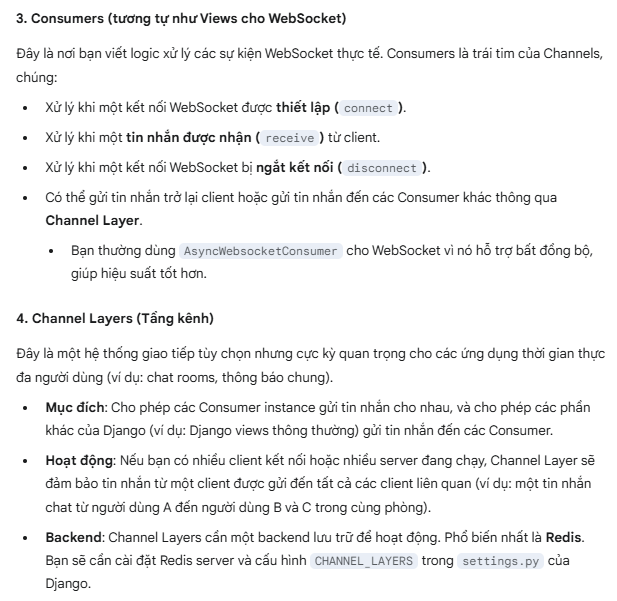
- Cấu hình ***ProtocolTypeRouter*** trong asgi.py:   


+ Nếu Asgi xác định request là http thì nó chuyển giao cho get\_asgi\_application() thông thường của Django

+ Nếu nó xác định là 1 request websocket nó sẽ chuyển giao cho AuthMiddlewareStack và UrlRouter của Channels để dẫn tới đúng **Consumer websocket**







2. tìm hiểu pythonsoketIO server (có cả client)

**3. resAPI được xử dụng để tích hợp websocket cùng với nhau mà không cần dùng thư viện trung gian như messageQueue?**

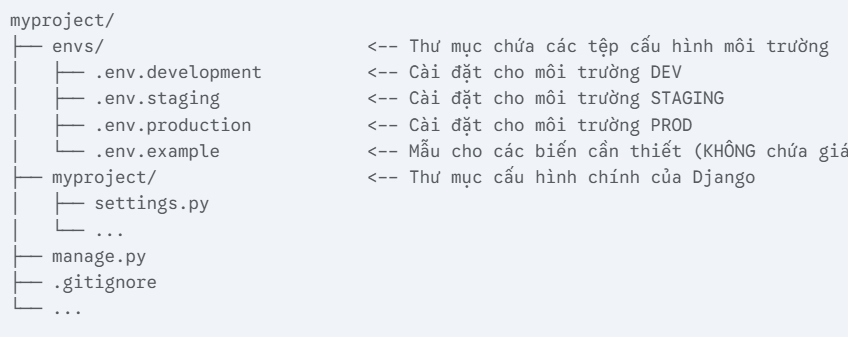
MessageQueue (Hàng đợi tin nhắn): là một cơ chế trong kiến trúc phần mềm, sử dụng để truyền tin nhắn dưới dạng massage giữa các thành phần khác nhau của hệ thống theo cơ chết bất đồng bộ. Nó như một bộ đệm trung gian được lưu trữ các massage để chờ xử lí.

- WorkFlow: Một Producer sẽ tạo ra massage và gửi tới MQ, nó sẽ không đợi phản hồi ngay lập tức mà tiếp tục thực hiện việc khác; MQ lưu trữ tin nhắn thường dưới dạng hàng đợi FIFO; Một Comsumer kết nối tới MQ nhận tin nhắn từ hàng đợi và bắt đầu xử lí logic, sau khi xử lí xong massage bị xóa khỏi hàng đợi.

- Một số công nghệ MQ phổ biến: RabbitMQ, Apache Kafka, Redis

4. pyENV quản lí virtual env như nào? Cách tách biệt project và file env.

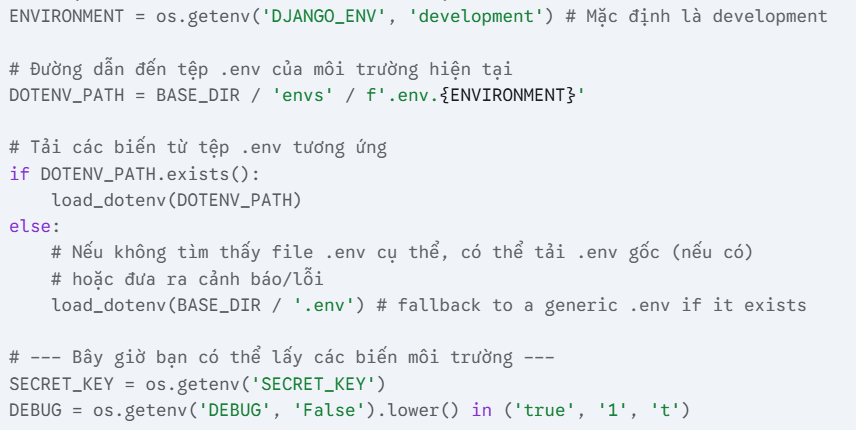
**5. cần nhiều file env cho các môi trường(môi trường dev staging, testing, production) được lưu vào folder envs của dự án trong đó chứa commonENV và các envs khác cho từng môi trường**

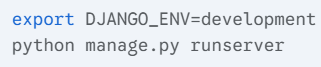
**-** Trong thực tế cần tạo folder envs chứa các file envs khác nhau phục vụ cho các môi trường khác nhau  


- cài đặt thư viện dotenv để load dữ liệu từ file .env:

pip install python-dotenv

- Sử dụng biến ENVIREMENT trong setting.py để chỉ định môi trường đang làm việc cùng



- Sau đó chỉ định tên môi trường cho biến ENVIROMENT trước khi run server:   




6. Tìm hiểu cách dùng swagger với restframework

7. Tìm hiểu intergration test(tạo db tới duyệt qua api, trả về respond...) và mock test - unittest (test các func, unnit nhỏ)

**8. built-in view của restframework (viewset; apiview; generateapiview - mỗi tầng sẽ có một mức độ detail khác nhau)**

- APIView: sử dụng định nghĩa các logic phức tạp, tùy chỉnh cao. Tự định nghĩa và khởi tạo QuerySet và Serializers

- GenericAPIView: Không định nghĩa phương thức Http nào giống như APIView, tuy nhiên nó đã định nghĩa rõ ràng model và serializer mà view sẽ làm việc, cung cấp các phương thức để lấy queryset, lấy đối tượng cụ thể, hoặc khởi tạo serializer một cách nhất quán, cho phép bạn cấu hình các hành vi như phân trang (pagination), lọc (filtering), tìm kiếm (searching) ở một nơi.

**Các phương thức quan trọng:** get\_queryset(), get\_serializer(), get\_object()

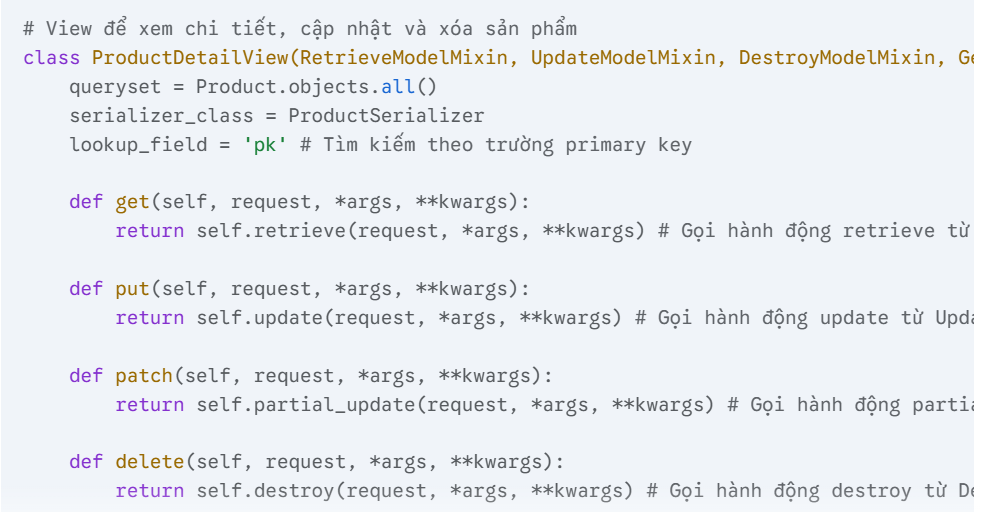
**Các thuộc tính quan trọng:**

* **queryset** (bắt buộc): chỉ định model view sẽ làm việc
  + queryset = Product.objects.all()
* **serializer\_class** (bắt buộc): chỉ định lớp serializer mà view sẽ sử dụng để chuyển đổi dữ liệu từ/sang model
  + Ví dụ: serializer\_class = ProductSerializer
* **lookup\_field**: Chỉ định tên trường trong URL mà view sẽ sử dụng để tìm kiếm một đối tượng cụ thể (mặc định là pk - primary key)
  + lookup\_field = 'slug' (tìm kiếm bằng field = ‘slug’)
* **lookup**\_url\_kwarg
* **filter\_backends**: danh sách các lớp backend bộ lọc được sử dụng để lọc queryset
* **pagination\_class**: lớp phân trang sử dụng cho view này

Trong thực tế **GenericAPIView** luôn được sử dụng cùng với các View **Mixins** để định nghĩa các HTTP handler:

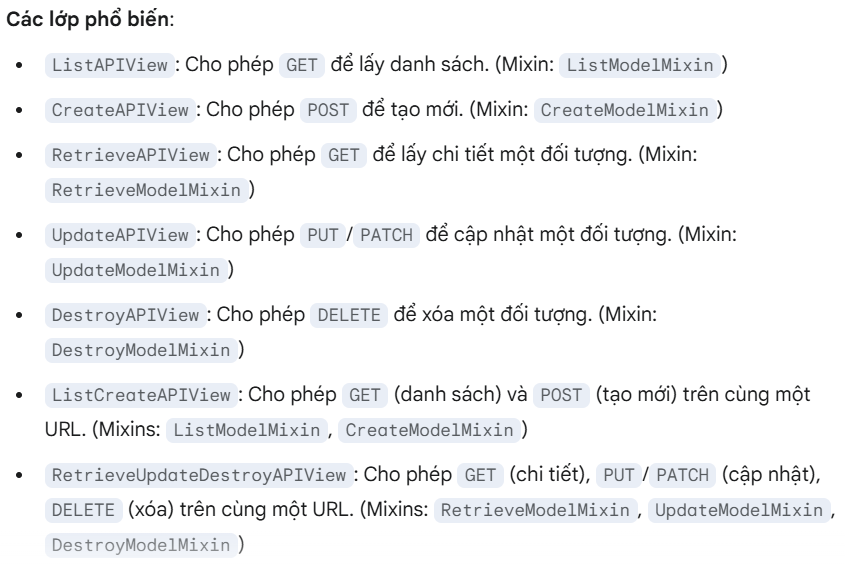
* **ListModelMixin**: cung cấp hành động lấy danh sách từ các đối tượng queryset. Ghi đè **list()**
* **CreateModelMixin**: tạo đối tượng từ model. Ghi đè **create()**
* **RetrieveModelMixin**: Lấy một đối tượng model. Cần ghi đè phương thức **retrieve**
* **UpdateModelMixin**: cập nhật đối tượng. ghi đè **update()/ partial\_update()**
* **DestroyModelMixin**: Xóa đối tượng. ghi đè **destroy()**

****



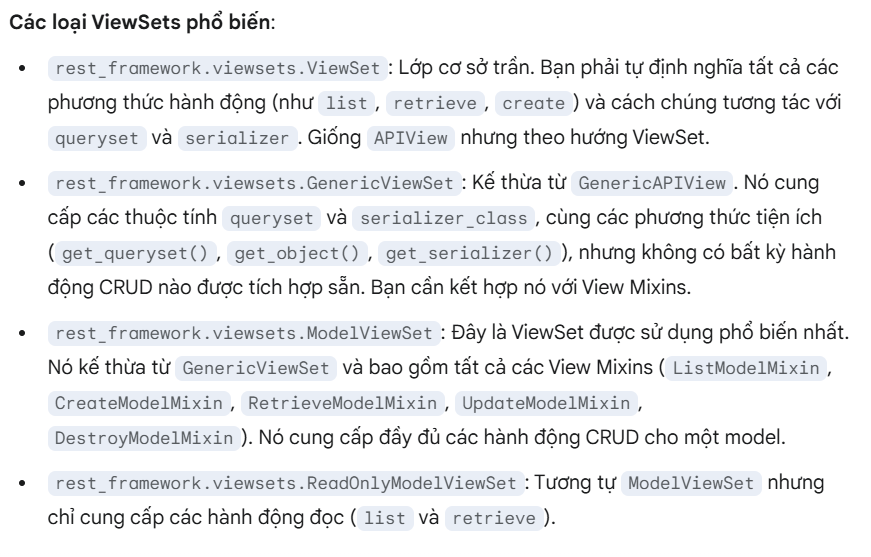
* Concrete View Classes: là GenericAPIView kết hợp với Mixin được định nghĩa hoàn chỉnh cho các tác vụ như lấy danh sách, tạo mới, lấy chi tiết, cập nhật, hoặc xóa đối tượng.

Là các lớp view độc lập, mỗi lớp xử lý một hoặc một vài thao tác CRUD cụ thể trên **một URL duy nhất**; Đã có sẵn phương thức HTTP handlers; chỉ cần khai báo queryset và serializer\_class



* **ViewSets:** Thay vì định nghĩa các hành động cho các phương thức HTTP (get, post), ViewSets định nghĩa các hành động cho một **tập hợp các hoạt động trên một tài nguyên**

Nhóm logic cho toàn bộ một tài nguyên vào một lớp duy nhất; Không có phương thức HTTP handlers (get, post, v.v.) trực tiếp. Thay vào đó, chúng có các phương thức hành động (list(), create(), retrieve(), update(), destroy(), v.v.); sử dụng với DefaultRouter hoặc SimpleRouter để tự động định tuyến



|  |  |
| --- | --- |
| Tên lớp View | Mô tả |
| APIView | Đây là lớp view cơ bản nhất, cung cấp một khung sườn linh hoạt để bạn tự định nghĩa logic cho từng HTTP method (GET, POST, PUT, DELETE, etc.). Nó không đi kèm với bất kỳ logic CRUD nào được tích hợp sẵn. |
| GenericAPIView | Mở rộng từ APIView, lớp này cung cấp các thuộc tính và phương thức hữu ích để làm việc với các queryset và serializer. Nó là nền tảng cho các lớp view hỗn hợp (mixins) và bộ view (viewsets). |
| Các View Mixin | Đây là các lớp bổ trợ, thường được sử dụng kết hợp với GenericAPIView để thêm logic xử lý các tác vụ CRUD cụ thể. Ví dụ: ListModelMixin cho phép liệt kê các đối tượng, RetrieveModelMixin cho phép lấy một đối tượng cụ thể. |
| Concrete View Classes | Đây là các lớp view hoàn chỉnh, kết hợp GenericAPIView với một hoặc nhiều mixin để tạo ra các view sẵn sàng sử dụng cho các tác vụ CRUD phổ biến. Ví dụ: ListCreateAPIView kết hợp ListModelMixin và CreateModelMixin để xử lý cả việc liệt kê và tạo đối tượng. |
| ViewSets | ViewSets nhóm logic cho một tập hợp các view liên quan vào một lớp duy nhất. Thay vì viết các view riêng biệt cho liệt kê và chi tiết, bạn có thể định nghĩa tất cả các hành động này trong một ViewSet duy nhất, sau đó sử dụng Router để tự động tạo URL cho chúng. ModelViewSet là một ví dụ phổ biến, xử lý đầy đủ các thao tác CRUD. |

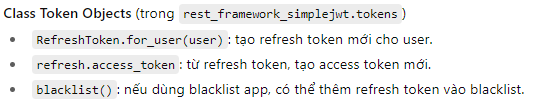
**9. token lưu trong sessionstorage, localstorage, cookie khác nhau thế nào? khi nào thì lưu ở đâu**

- sessionStorage: lưu trong thời gian ngắn theo phiên, mất khi đóng tab/đóng trình duyệt

- localStorage: lưu trong thời gian dài hơn session, chỉ mất khi xóa thủ công hoặc bằng code. Dữ liệu vẫn còn sau khi đóng tab/ trình duyệt

Nhược điểm của 2 cách trên: sessionStorage mất login khi đóng ứng dụng, kém tiện dụng. Cả 2 cách trên đều dễ bị tấn công XSS. Nếu muốn đăng nhập lâu dài và bảo mật cao có thể dùng **HTTPOnly Secure Cookie,** còn muốn tiện lợi có thể dùng **localStorage + refresh token**

***Các methods của RefreshToken***

****

**9.1. LocalStorage + refresh token:**

- Hạn chế thời gian sống của refresh token

- Dùng token rotation: mỗi lần sử dụng refresh token thì tạo refresh token mới và vô hiệu cái cũ

**- CSP (Content Security Policy)** nghiêm ngặt để hạn chế chèn script

**- Escape dữ liệu đầu vào** để giảm nguy cơ XSS

**9.2.** **HTTPOnly Secure Cookie**

**-** Dùng HttpOnly để **trình duyệt chặn mọi truy cập từ JS tới cookie**. Cookie chỉ được gửi kèm request HTTP/HTTPS.

- **Secure** – Chống MITM (Man-in-the-Middle) nghe lén: Dùng Secure để cookie **chỉ được gửi khi kết nối HTTPS** (TLS/SSL). Dữ liệu sẽ được mã hóa trên đường truyền, kẻ tấn công không đọc được.

- sử dụng SameSite=Strict khi cấu hình cookie để giảm nguy cơ **CSRF** (Cross-Site Request Forgery)

**XSS (Cross-Site Scripting):** Là một dạng tấn công web chèn mã JS vào mã hợp lệ của web thông qua form url comment và người dùng sẽ thực thi các script đó và bị đánh cắp cookie, token, chiếm quyền phiên,…

**MITM (man in the middle):** người tấn công đứng giữa client và server chen ngang kênh giao tiếp nhằm thay đổi request/respond hoặc đánh cắp thông tin (sniffing)

**10. tìm hiểu cụ thể cấu tạo token, cách server kiểm tra validate**

**- Cơ chế validate** trong Django: sau khi người dùng đăng nhập thành công, mật khẩu được hash 1 chiều sử dụng 1 thuật toán hash + số lần hash + salt(kết hợp với password để hash) -> tạo ra mật khấu đã hash với cấu trúc: tên algHash\_số lần hash\_salt\_password sau khi hash

Sau đó khi user đăng nhập cung cấp username và password(plaintext). Sauk hi kiểm tra username đúng nó sẽ lấy mật khẩu đã hash tương ứng, dựa vào thông tin đã hash, nó sẽ hash lại password mà người dùng cung cấp và so sánh với mật khẩu đã hash trong DB -> nếu giống nhau -> validated

**- Cấu tạo của token**

**- Header:** dạng Json chứa thông tin alg – thuật toán hash được sử dụng để kí (sign) JWT

- Payload: dạng Json chứa dữ liệu khẳng định (claims) về thực thể (người dùng) và các dữ liệu bổ sung. Có 3 loại claims: registered, public, private

- Signature: phần quan trọng nhất đảm bảo toàn vẹn và xác thực của JWT. SecretKey được lưu ở server

Signature = HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

JWT = Header (base64Url) . Payload (base64Url) . Signature

**11. Cơ chế hoạt động của refresh token**

**-** refresh token: sử dụng để lấy access token mới mà không cần đăng nhập lại khi token cũ hết hạn

- quy trình: khi đăng nhập với access token hết hạn -> API trả về 401 Unauthor hoặc token expired -> client tự gọi tới /refresh kèm refresh token (nếu dùng HTTPOnly → browser tự gửi cookie/ dùng ocalStorage/sessionStorage → client tự đọc và gửi) -> server xác thực refresh token -> nếu hợp lệ và còn sống tạo access token mới và refresh rotation(tùy cấu hình) -> trả lại cho client

Trong trường hợp refresh token hết hạn -> redirect về login và người dùng cần đăng nhập lại.

12. chức năng của nginx, gunicorn, cấu hình nginx cơ bản

13. cơ chế khi request gửi tới nginx, guinicorn, nó tạo thread hoặc process như nào? blogDB

thread process ảnh hưởng tới db như nào? có bị block db và static global variable conflict không?

**14. cách để tránh n+1 problem trong DB**

- Là vấn đề khi thực hiện truy vấn CSDL để lấy thông tin nào đó mà dùng tới N+1 query để lấy dữ liệu của N items.

- Trong Django có thể sử dụng các biện pháp sau nhằm tránh N+1 problem:  
 - Đối với các truy vấn OneToOne hoặc là ForeignKey(FK) thì sử dụng select\_related() với QuerySet. **Nó thực hiện một truy vấn JOIN trong cơ sở dữ liệu (bảng chứa khóa phụ và bảng gốc)**

- Đối với các truy vấn là ManyToMany hoặc là mqh ngược của ForeignKey sử dụng fetch\_related(). Nó sẽ thực hiện 2 truy vấn riêng biệt và ghép nối với nhau trong bộ nhớ. 15.trong các ứng dụng phức tạp thì phần logic chính được tách dời viết trong service.py, sau đó view gọi tới service

16.tạo chat app

**Các thông tin thêm**

**Các options khi tạo field:**

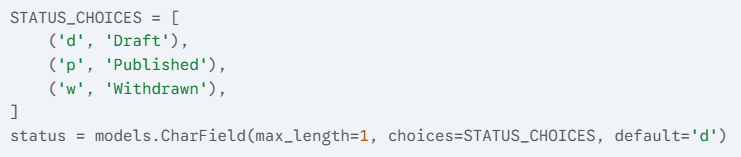
Null = true/false

Blank= true/false – để trống hoặc phải ghi thông tin của trường vào form đăng ký.

Default = value/callable func

unique=True/false – trường này có giá trị duy nhất đối với từng giá trị

choices

****

help\_text: đoạn văn bản trợ giúp ngắn gọn để hiển thị trong biểu mẫu Django (chủ yếu trong Django Admin) hoặc các biểu mẫu tự tạo

verbose\_name: Cung cấp một tên thân thiện, dễ đọc cho trường, được sử dụng trong Django Admin và các thông báo lỗi. Mặc định ví dụ full\_name -> Full Name trong admin

primary\_key = true/false

db\_index = true/false – tạo chỉ mục cho csdl này trog db

editable = true/false - Kiểm soát liệu trường có thể được chỉnh sửa trong Django Admin hoặc trong các model form mặc định hay không.

**Các trường dữ liệu trong Meta khi khai báo class models:**

1. Verbose\_name ; verbose\_name\_plural

2. ordering: ordering = ['category', 'title'] – khai báo thứ tự record khi truy vấn (tương đương với thêm **ORDER BY category, title** với bất kì lệnh truy vấn nào trong model)

3. db\_table – chỉ định tên trong csdl

4. unique\_together = (('author', 'title'),) – chỉ định dữ liệu trong các cặp được khai báo là duy nhất

5. indexes – tạo chỉ mục csdl. Nó định nghĩa một cấu trúc chỉ mục dạng B-tree chứa các địa chỉ trỏ tới các record của 1 trường dữ liệu được định nghĩa.

Ví dụ: **models.Index(fields=['is\_published', '-published\_date'])**

Đây là một **chỉ mục kết hợp** (composite index). Nó được sắp xếp dựa trên cả hai trường. Điều này cực kỳ hữu ích cho các truy vấn vừa lọc vừa sắp xếp, chẳng hạn như **WHERE is\_published = True ORDER BY published\_date DESC**

6. abstract = True/False – không tạo bảng trong csdl; dùng để lớp khác kế thừa và tùy chỉnh Meta

7. app\_label – chỉ định tên ứng dụng thuộc về (ít dùng)

**Tạo form: Chỉ sử dụng với các dự án có FE và BE dính liền (SSR – server side redering). Nếu dùng FE và BE tách dời thì cần xử lí thủ công dùng serializers của DRF (kiến trúc SPA - Single Page Application)**

* + 1. **UserCreationForm:** kế thừa từ ModelForm, được Django cung cấp, nó được cấu hình để làm việc với User mặc định của Django.

Chỉ cần kế thừa và tùy chỉnh các trường cần thiết

Tích hợp các logic để xử lí mật khẩu an toàn(hash validation); code ít

Ít linh hoạt chủ yếu tạo người dùng mới

Class Meta trong UserCreationForm: được cấu hình sẵn để xử lí các trường cần thiết cho việc đăng kí người dùng mới

Model: tên model

Fields: mặc định đã có trường username và password

* + 1. **ModelForm:** form đa năng trong Django, tự động tạo các trường form dựa trên các trường của một model đã có

Linh hoạt: tính tùy chỉnh cao, phù hợp cho cả tạo mới và thay đổi đối tượng model

Cần tự thêm logic validation mật khẩu và hash mật khẩu an toàn

Class Meta trong ModelForm:

**Model** – bắt buộc , tên của Model đang làm việc

**Fields** – liệt kê các trường đưa vào danh sách ( bằng 1 list or 1 tuple)

**Exclude** – loại bỏ các trường không cho vào form (cách an toàn khi không muốn hiển thị trường nhạy cảm)

**labels**: Tùy chỉnh nhãn hiển thị cho các trường. Ví dụ: labels = {'full\_name': 'Họ và Tên'}.

**widgets**: Cung cấp widget HTML tùy chỉnh cho một trường

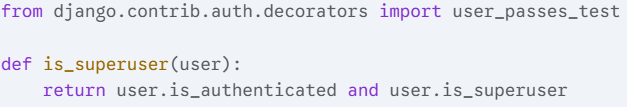
**help\_texts**: Cung cấp văn bản trợ giúp cho các trường.

**Decorators trong việc xác thực(authentication) và quyền truy cập(permission) vào các view**

1. **Phân quyền**
   * **@login\_required**: đảm bảo người dùng đã đăng nhập. Nếu chưa thì chuyển hướng tới settings.LOGIN\_URL
   * **@permission\_required**: đảm bảo đã đăng nhập và có quyền cụ thể để truy cập vào view. Có thể truyền vào một chuổi tên quyền. Nếu không có đủ quyền thì nó chuyển hướng tới trang đăng nhập hoặc trang báo lỗi

**Ví dụ:** @permission\_required(('blog.can\_edit\_article', 'blog.can\_delete\_article'))

* **@staff\_member\_required:** đảm bảo người dùng đã đăng nhập và có is\_staff = True.Thường dành cho nhân viên nội bộ hoặc quản trị viên
* **@superuser\_required** (Không có sẵn, thường tự tạo)

****

* **@user\_passes\_test(test\_func, login\_url=None, redirect\_field\_name=REDIRECT\_FIELD\_NAME)**

Đây là decorator linh hoạt nhất, kiểm tra bất kì điều kiện tùy chỉnh nào đối với người dùng

* **@csrf\_exempt**: Miễn trừ một view khỏi CSRF protection
* **@csrf\_protect**: Buộc CSRF protection cho một view.

1. **Quyền truy cập (permission) sử dụng trong CBV:** trong DRF xử dụng permission thay cho decorators (decorators vẫn dùng được cho FBV)
   * **IsAuthenticated**: Tương tự @login\_required.
   * **IsAdminUser**: Tương tự @staff\_member\_required (kiểm tra is\_staff=True).
   * **IsAuthenticatedOrReadOnly**: Cho phép người dùng đã xác thực thực hiện mọi hành động, nhưng người dùng chưa xác thực chỉ có thể đọc.
   * **DjangoModelPermissions / DjangoObjectPermissions**: Để kiểm tra các quyền dựa trên model hoặc đối tượng cụ thể.

**Nguyên tắc RestFul (Representational State Tranfer)**

**Là một phòng cách kiến trúc để thiết kế hệ thống nối mạng**

* 1. Client – server: tách biệt người dùng client (FE) và server (BE)
  2. Stateless – không trạng thái: mỗi request phải chứa tất cả thông tin để server hiểu yêu cầu đó và đưa ra phản hồi, server không lưu trữ session nào của client
  3. Cacheable: Phản hồi từ server được xác định rõ xem client có cache hay không
  4. Layered system (Hệ thống phân tầng): client không cần biết nó đang kết nối với server hay là một lớp trung gian
  5. Uniform interface (giao diện thống nhất)
     + Tài nguyên được xác định bằng url
     + Client gửi biểu diễn tài nguyên (JSON) để thay đổi thông tin
     + Mỗi thông báo chứa đủ thông tin để client xử lí nó
     + Client tương tác với server thông qua các Url (các components trong giao diện) nhận được từ phản hồi (phản hồi trả về trang homepage sau đó có thể điều hướng đi các trang khác)
  6. Code-on-demand: server có thể tạm thời mở rộng hoặc tùy chỉnh các chức năng của client thông qua mã thực thi

Tác dụng và các loại validate trong **serializers; Các built-in serizlizers**

* + - Tác dụng:
      * Serialization và deserialization
      * Xác thực (validation): đảm bảo dữ liệu đầu vào là hợp lệ trước khi sử dụng để tạo hoặc cập nhật đổi tượng. Việc này giúp **tách biệt mối quan tâm** (model chỉ chứa logic liên quan tới DB, view chỉ chứa logic liên quan tới request/respond, phân quyền và điều phối)
    - Có 2 loại validate trong serializers:
      * Xác thực cấp trường (validate\_filed\_name): kiểm tra tính hợp lệ của 1 trường
      * Xác thực cấp đối tượng (validate): kiểm tra tính hợp lệ của nhiều tường cùng lúc hoặc ràng buộc giữa chúng

- Các loại Serializers:

1. HyperlinkedModelSerializer

Tác dụng:

Thay vì trả về ID của các đối tượng liên quan, nó trả về URL của object đó.

Thích hợp khi xây dựng API theo RESTful với liên kết giữa resources.

2.ListSerializer

Tác dụng:

Dùng để serialize mảng object, hỗ trợ batch operations như create/update nhiều object cùng lúc.

Thường sử dụng nội bộ khi bạn muốn customize serialization cho danh sách.

3. SerializerMethodField

Tác dụng:

Tạo các trường tính toán, không lưu vào database.

Thường dùng để kết hợp nhiều field, format dữ liệu, hoặc thêm dữ liệu từ các nguồn khác.

4. Custom Serializer Fields

Tác dụng:

Tạo các field tùy chỉnh với logic validate, format hoặc representation riêng.

5. Relation Fields

Tác dụng:

Dùng để serialize các trường quan hệ (ForeignKey, ManyToManyField) theo cách mong muốn.

Các loại:

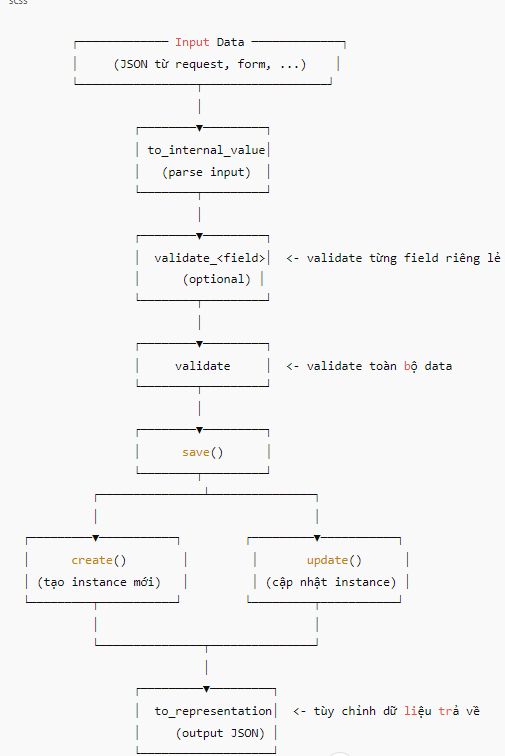
PrimaryKeyRelatedField: trả về ID của object liên quan.

StringRelatedField: trả về kết quả \_\_str\_\_() của object liên quan.

SlugRelatedField: trả về giá trị của một field cụ thể (slug) thay vì ID.

**Các hàm sử dụng trong và luồng workflow thực hiện trong serializer:**

****



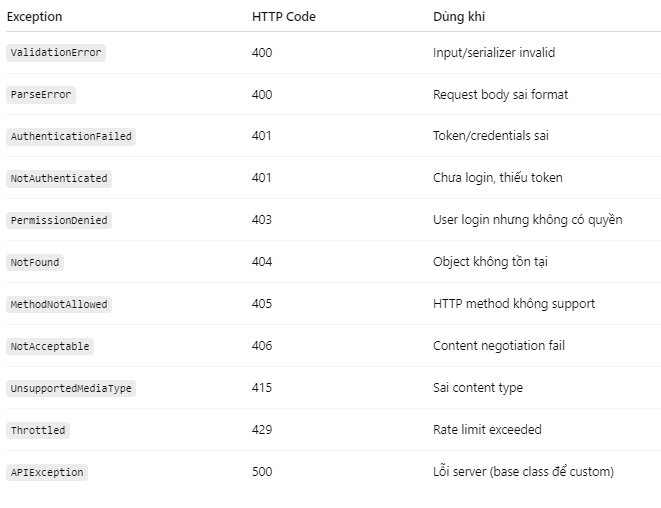
**Các loại Cookie**

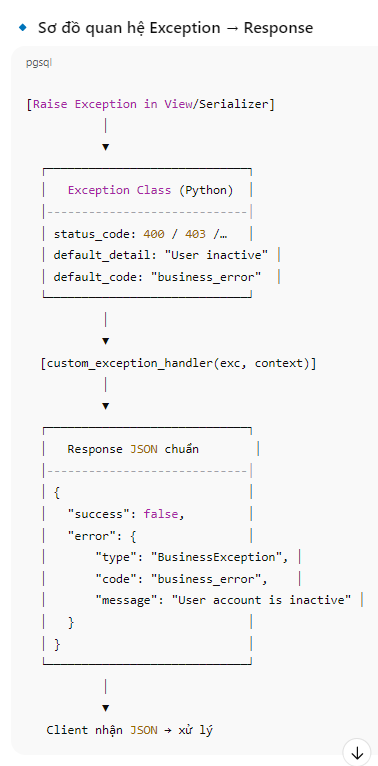
Theo thời gian tồn tại: Session cookies/ Persistent Cookies

Theo nguồn gốc: First-party cookies/ Third-party cookies

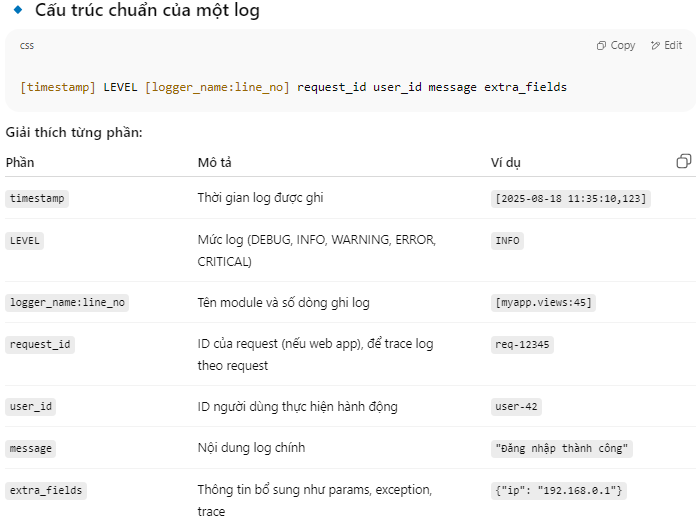
Theo bảo mật: Secure cookies/ Cookie HttpOnly/ Cookie SameSite

**Exceptions trong DRF:**

****



**Config Logging DRF**

****

**Các loại log level:**

****