🔹 Utilisation de `get_object` dans une vue générique Django REST Framework (DRF)

Dans **Django REST Framework (DRF)**, la méthode `get_object` est utilisée dans les **vues génériques** pour récupérer un objet unique correspondant à une requête donnée.

🔹 Pourquoi utiliser `get_object` ?

Par défaut, les vues génériques comme `RetrieveAPIView`, `UpdateAPIView` ou `DestroyAPIView` utilisent `get_object` pour **trouver l'objet à manipuler**.

📌 **Cas d'utilisation :**

✅ Récupérer un objet via son `pk` ou un autre champ (`lookup_field`)
✅ Filtrer les objets selon l'utilisateur connecté
✅ Gérer des permissions spécifiques avant la récupération

🔹 Exemple 1 : Récupérer un objet avec le `lookup_field` (par défaut `pk`)

Dans une vue de récupération (`RetrieveAPIView`), `get_object` est utilisé pour trouver un objet selon `pk` (valeur par défaut).

```
from rest_framework.generics import RetrieveAPIView
from myapp.models import Article
from myapp.serializers import ArticleSerializer

class ArticleDetailView(RetrieveAPIView):
    queryset = Article.objects.all()
    serializer_class = ArticleSerializer
```

📌 **Requête :**

```
GET /articles/3/
```

➡ **DRF appelle `get_object`** pour récupérer l'article ayant `pk=3`.

🔹 Exemple 2 : Utiliser un `lookup_field` différent de `pk`

Si on veut récupérer un article avec un champ **slug** au lieu du `pk` :

```python
class ArticleDetailView(RetrieveAPIView):
    queryset = Article.objects.all()
    serializer_class = ArticleSerializer
    lookup_field = "slug"
```

📌 **Requête :**

```
GET /articles/django-introduction/
```

➡ **DRF cherche un article avec `slug="django-introduction"` au lieu de `pk`.**

🔹 Exemple 3 : Personnaliser `get_object` (Filtrer par utilisateur connecté)

Si chaque utilisateur ne peut voir que ses propres articles, on peut personnaliser `get_object` :

```python
class ArticleDetailView(RetrieveAPIView):
    serializer_class = ArticleSerializer

    def get_queryset(self):
        # Récupérer uniquement les articles de l'utilisateur connecté
        return Article.objects.filter(author=self.request.user)

    def get_object(self):
        # On appelle la version standard de get_object() mais sur notre queryset filtré
        queryset = self.get_queryset()
        obj = super().get_object()

        # Vérification supplémentaire (facultatif)
        if obj.author != self.request.user:
            raise PermissionDenied("Vous n'avez pas accès à cet article.")

        return obj
```

📌 **Requête :**

```
GET /articles/2/  (Utilisateur A)
```

✅ **Si l'article 2 appartient à l'utilisateur A → accès autorisé**
❌ **Sinon → `403 Forbidden`**

🔹 Exemple 4 : `get_object` avec `lookup_field` dynamique

Si on veut que l'API supporte **soit un `pk`, soit un `slug`** :

```
class ArticleDetailView(RetrieveAPIView):
    serializer_class = ArticleSerializer

    def get_object(self):
        queryset = Article.objects.all()
        lookup_value = self.kwargs.get("pk_or_slug")  # On récupère le
paramètre dynamique

        if lookup_value.isnumeric():
            return queryset.get(pk=lookup_value)  # Rechercher par `id`
        return queryset.get(slug=lookup_value)  # Rechercher par `slug`
```

📌 **Requêtes possibles :**

```
GET /articles/3/       (Recherche par `id`)
GET /articles/django-intro/ (Recherche par `slug`)
```

🔹 Résumé

Méthode	Rôle
`get_object`	Récupère un objet unique selon `lookup_field` ou logique personnalisée
Cas d'utilisation	- Filtrer un objet selon `request.user` - Gérer des permissions spécifiques - Rechercher via `slug` ou `pk` dynamiquement

📌 **Par défaut ?** ✅ Oui, mais personnalisable c

📌 **Obligatoire ?** ❌ Non, DRF l'appelle automatiquement dans les vues génériques

🚀 **`get_object` est clé pour gérer la récupération sécurisée des objets !**