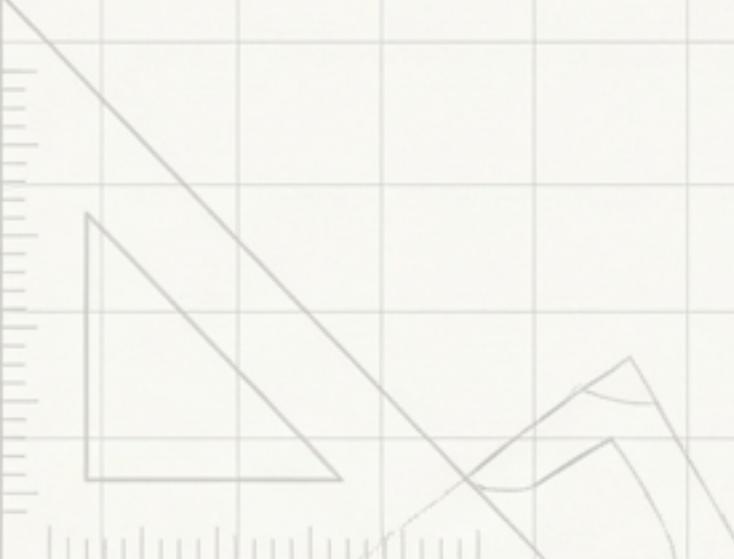


تسلط بر Generic View ها در Django REST Framework

یک راهنمای جامع و دقیق بر اساس معماری و مستندات رسمی



فلسفه اصلی: چرا Generic View ها به وجود آمدند؟

«Generic view های جنگو به عنوان یک میانبر برای الگوهای استفاده‌ی رایج توسعه داده شدند... آن‌ها اصطلاحات و الگوهای متداول در توسعه view را انتزاعی می‌کنند تا بتوانید به سرعت view های رایج داده را بنویسید، بدون آنکه مجبور به تکرار خود باشید.» - مستندات جنگو

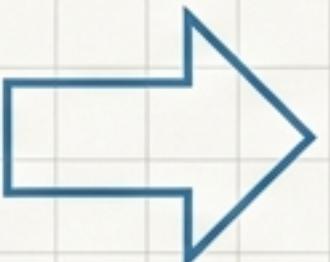
مزیت اصلی Generic View ها در DRF، انتزاعی کردن الگوهای رایج (مانند لیست کردن، ایجاد، بازیابی، آپدیت و حذف اشیاء) است. این کار به ما اجازه می‌دهد کدی بسیار خلاصه‌تر و قابل استفاده مجدد بنویسیم.

قبل: پیاده‌سازی دستی با `APIView`

```
from rest_framework.views import APIView
from rest_framework.response import Response
from myapp.models import MyModel
from myapp.serializers import MyModelSerializer

class MyModelList(APIView):
    def get(self, request, format=None):
        queryset = MyModel.objects.all()
        serializer = MyModelSerializer(queryset, many=True)
        return Response(serializer.data)

    def post(self, request, format=None):
        serializer = MyModelSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=201)
        return Response(serializer.errors, status=400)
```



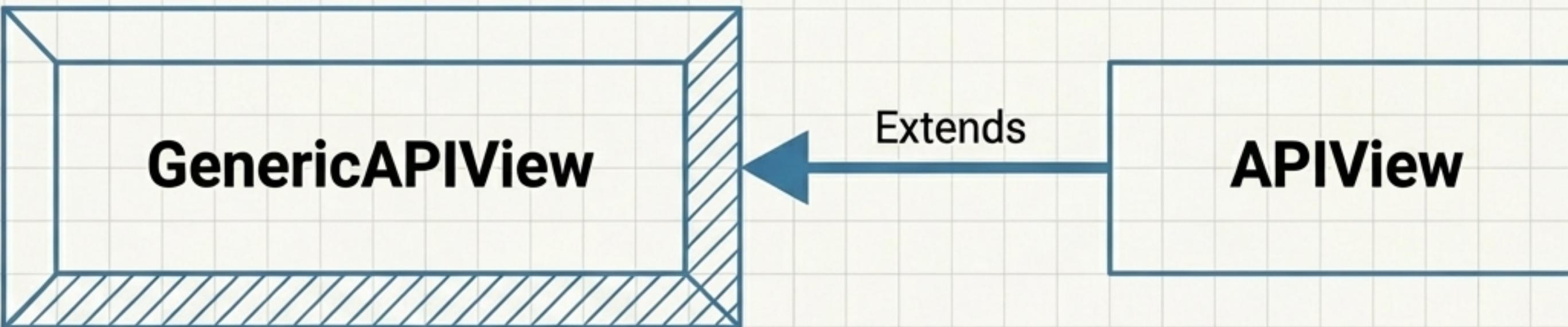
بعد: استفاده از `generics.ListCreateAPIView`

```
from rest_framework import generics
from myapp.models import MyModel
from myapp.serializers import MyModelSerializer

class MyModelList(generics.ListCreateAPIView):
    queryset = MyModel.objects.all()
    serializer_class = MyModelSerializer
```

شالوده: معرفی `GenericAPIView`

نکته کلیدی: این کلاس، نقطه‌ی شروع و قلب تپنده‌ی تمام View ها است.



- DRF خود APIView کلاس GenericAPIView را توسعه می‌دهد.
- این کلاس به تنها‌یی متدهای `()`, `get()`, `post()` و غیره را پیاده‌سازی نمی‌کند.
- وظیفه‌ی اصلی آن، فراهم کردن رفتارهای هسته‌ای و مشترک برای کار با مدل‌ها است، مانند تعیین `queryset` و `.serializer`.
- تمام view‌های آماده‌ی دیگر با ترکیب این کلاس و یک یا چند Mixin ساخته می‌شوند.

بلوپرینت `GenericAPIView`: پیکربندی‌های اصلی (Attributes)

این خصوصیات، رفتار اصلی view را کنترل می‌کنند.

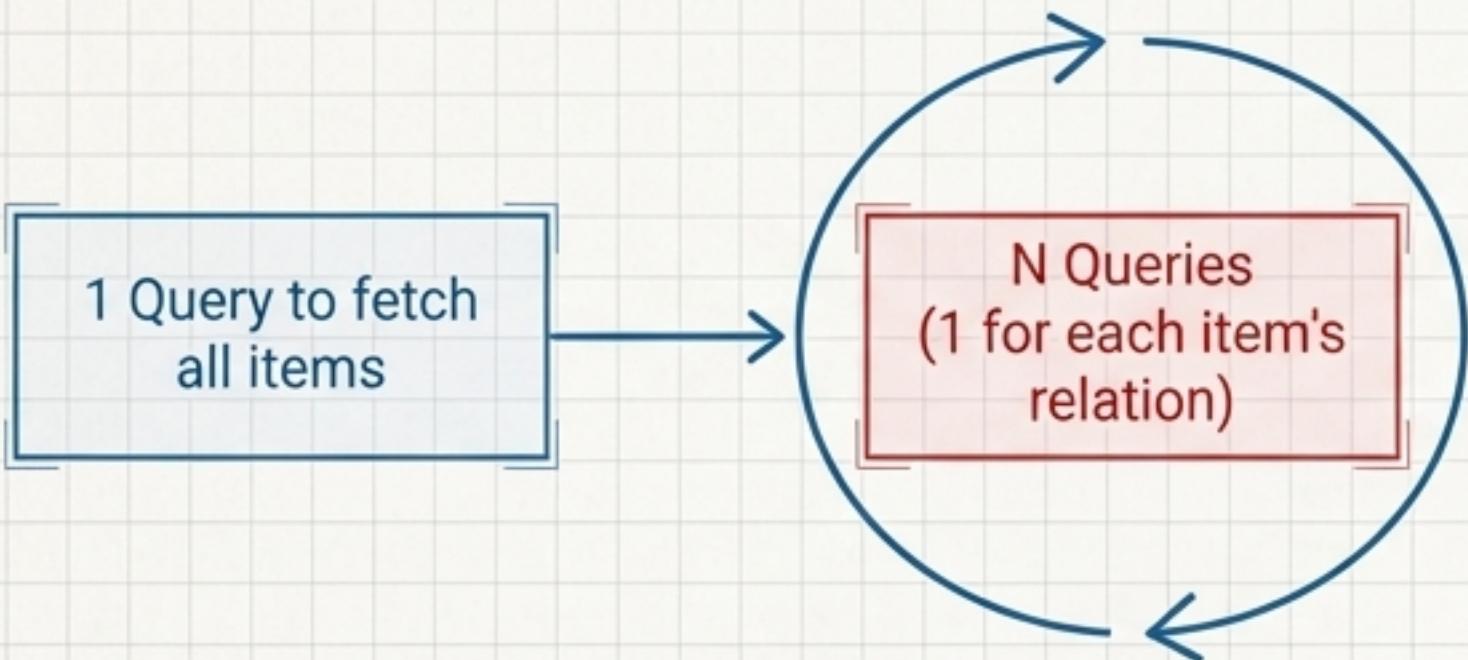
خاصیت	توضیح
	بخش: تنظیمات پایه (Basic Settings)
queryset	کوئری‌ستی که برای بازگرداندن اشیاء استفاده می‌شود. باید این خصوصیت را تنظیم کنید یا متدهای override() را get_queryset() نمایید.
serializer_class	کلاس سریالایزری که برای اعتبارسنجی ورودی و سریالایز کردن خروجی استفاده می‌شود.
lookup_field	فیلد مدلی که برای پیدا کردن یک نمونه‌ی خاص از مدل استفاده می‌شود. مقدار پیش‌فرض آن 'pk' است.
lookup_url_kwarg	آرگومان کلیدی در URL که برای جستجوی آبجکت استفاده می‌شود. اگر تنظیم نشود، مقدار lookup_field را می‌گیرد.
	بخش: صفحه‌بندی (Pagination)
pagination_class	کلاسی که برای صفحه‌بندی نتایج لیست استفاده می‌شود. در صورت None بودن، صفحه‌بندی غیرفعال می‌شود.
	بخش: فیلترینگ (Filtering)
filter_backends	لیستی از کلاس‌های بکاند فیلتر که برای فیلتر کردن کوئری‌ست استفاده می‌شوند.

مکانیزم‌های اصلی (Methods) : `متدهای کلیدی` GenericAPIView

این متدها به شما اجازه می‌دهند رفتار view را به صورت داینامیک و بر اساس شرایط درخواست (request) تغییر دهید.

get_queryset(self)	get_object(self)	get_serializer_class(self)
<p>بازگرداندن کوئریست به صورت پویا. همیشه باید از این متدهای view برای دسترسی مستقیم به self.queryset استفاده کرد، زیرا queryset تنها یک بار ارزیابی و کش می‌شود.</p> <pre>def get_queryset(self): user = self.request.user return user.accounts.all()</pre>	<p>بازگرداندن یک نمونه‌ی خاص از مدل در view‌های جزئیات. می‌توان آن را برای جستجو بر اساس چندین پارامتر URL بازنویسی کرد.</p> <pre>def get_object(self): queryset = self.get_queryset() filter = {} for field in self.multiple_lookup_fields: filter[field] = self.kwargs[field] obj = get_object_or_404(queryset, **filter) self.check_object_permissions(self.request, obj) return obj</pre>	<p>انتخاب کلاس سریالایزر به صورت پویا. مثلاً برای کاربران مختلف یا عملیات خواندن/نوشتن، سرالایزرهای متفاوتی ارائه دهید.</p> <pre>def get_serializer_class(self): if self.request.user.is_staff: return FullAccountSerializer return BasicAccountSerializer</pre>

N+1 Queries: جلوگیری از مشکل



شرح مشکل:

هنگام لیست کردن اشیاء، اگر سریالایزر به روابط (relations) دسترسی پیدا کند، ممکن است به ازای هر آیتم یک کوئری اضافه به دیتابیس زده شود که عملکرد را به شدت کاهش می‌دهد.

ForeignKey & OneToOneField

راه حل: از `select_related` برای واکنشی آبجکت‌های مرتبط در همان کوئری اولیه استفاده کنید.

```
# For ForeignKey and OneToOneField
def get_queryset(self):
    # Joins tables in a single SQL query
    return Order.objects.select_related("customer", "billing_address")
```

Many-to-Many & Reverse Relations

راه حل: از `prefetch_related` برای بارگذاری بهینه‌ی مجموعه‌ای از آبجکت‌های مرتبط در یک کوئری جداگانه استفاده کنید.

```
# For reverse and many-to-many relationships
def get_queryset(self):
    # Performs a separate lookup, more
    # efficient for many relations
    return Book.objects.prefetch_related("categories", "reviews_user")
```

Combining Both

راه حل: برای سناریوهای پیچیده، هر دو متده را با هم ترکیب کنید.

```
# Combining both for complex scenarios
def get_queryset(self):
    return (Order.objects
        .select_related("customer")
        .prefetch_related("items_product")
    )
```

قلاب‌های ذخیره و حذف (Save and Deletion Hools)

این متدها توسط Mixin ها فراخوانی می‌شوند و به شما اجازه می‌دهند رفتار پیش‌فرض ذخیره یا حذف آبجکت را بازنویسی کنید. این‌ها نقاطی عالی برای تزریق منطق سفارشی هستند.

- هنگام ذخیره یک نمونه‌ی جدید فراخوانی می‌شود. CreateModelMixin : perform_create(self, serializer)
- هنگام ذخیره یک نمونه‌ی موجود فراخوانی می‌شود. UpdateModelMixin : perform_update(self, serializer)
- هنگام حذف یک نمونه فراخوانی می‌شود. DestroyModelMixin : perform_destroy(self, instance)

1. تنظیم داده‌های ضمنی
ذخیره‌ی کاربر فعلی بدون ارسال آن در درخواست.

```
def perform_create(self, serializer):
    serializer.save(user=self.request.user)
```

2. اجرای عوارض جانبی
ارسال ایمیل تایید پس از یک آپدیت موفق.

```
def perform_update(self, serializer):
    instance = serializer.save()
    send_email_confirmation(user=self.request.user, modified=instance)
```

3. اعتبارسنجی اضافی
بررسی یک شرط خاص قبل از ذخیره در دیتابیس.

```
def perform_create(self, serializer):
    if SignupRequest.objects.filter(user=self.request.user).exists():
        raise ValidationError('You have already signed up')
    serializer.save(user=self.request.user)
```

بخش دوم: قطعات سازنده: میکسین‌ها (Mixins) (Mixins)

میکسین‌ها: افزودن قابلیت‌های عملیاتی (Actions) (Actions)

میکسین‌ها متدهای عملیاتی (`.create()`, `.`.create(`...`), `.`.post(`.`.get(`.`)) را فراهم می‌کنند، اما به تنها یی متدهای پاسخگو به درخواست HTTP (مانند `.`.post(`.`) یا `.`.get(`.`)) را تعریف نمی‌کنند. این ویژگی به ما اجازه می‌دهد تا رفتارها را به صورت انعطاف‌پذیر با هم ترکیب کنیم.



کاتالوگ میکسین‌ها

این پنج میکسین، بلوک‌های اصلی سازنده‌ی تمام عملیات CRUD در Generic View ها هستند.

ListModelMixin

- متدها:
.list(request, ...)
- عملکرد: پیاده‌سازی لیست کردن یک کوئری است.
- پاسخ موفقیت: 200 OK

CreateModelMixin

- متدها:
.create(request, ...)
- عملکرد: پیاده‌سازی ایجاد و ذخیره یک نمونه‌ی جدید.
- پاسخ موفقیت: 201 Created

RetrieveModelMixin

- متدها:
.retrieve(request, ...)
- عملکرد: پیاده‌سازی بازگرداندن یک نمونه‌ی موجود.
- پاسخ موفقیت: 200 OK

UpdateModelMixin

- متدها:
.update(request, ...)
.partial_update(request, ...)
- عملکرد:
پیاده‌سازی آپدیت (PUT) و آپدیت جزئی (PATCH)
- پاسخ موفقیت: 200 OK

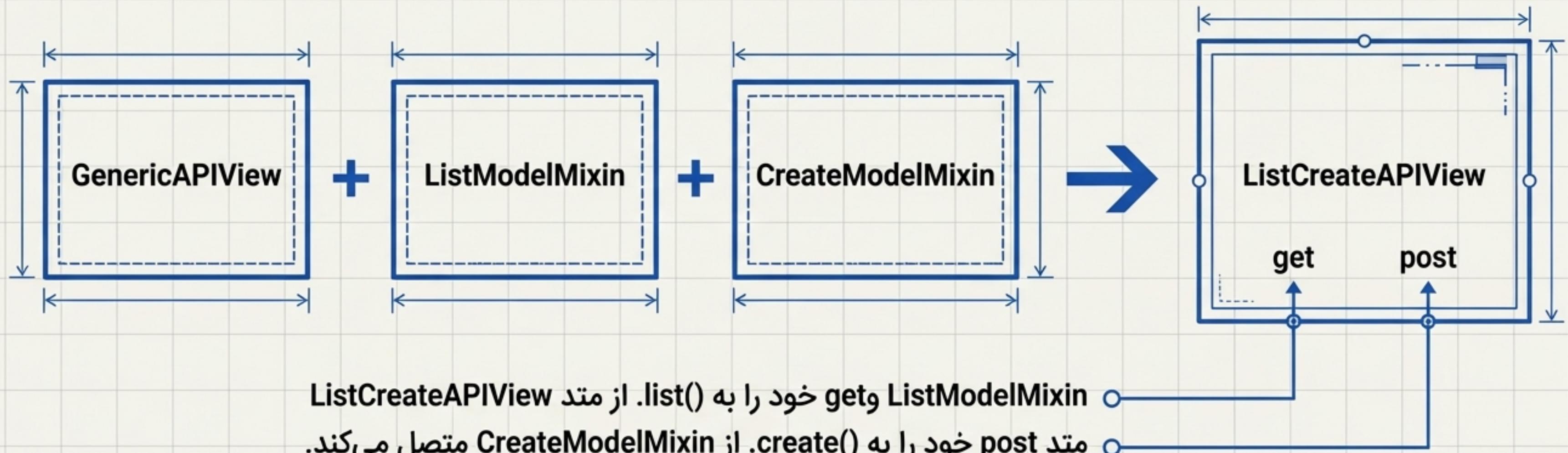
DestroyModelMixin

- متدها:
.destroy(request, ...)
- عملکرد:
پیاده‌سازی حذف یک نمونه‌ی موجود.
- پاسخ موفقیت: 204 No Content

بخش سوم: محصول نهایی: کلاس‌های آماده (Concrete View Classes)

کنار هم گذاشتن قطعات: تولد کلاس‌های View آماده

این کلاس‌ها، همان view‌هایی هستند که در عمل بیشترین استفاده را دارند. آن‌ها چیزی نیستند جز ترکیبی از GenericAPIView و یک یا چند Mixin. این کلاس‌ها متد‌های عملیاتی میکسین‌ها متصل می‌کنند.



مرجع کلاس‌های آماده (بخش اول: عملیات تکی)

این کلاس‌ها هر کدام یک یا دو عملیات اصلی را روی مجموعه‌ای از اشیاء یا یک شیء واحد انجام می‌دهند.

CreateAPIView

- کاربرد: فقط برای اندپوینت‌های ایجاد.(create-only)



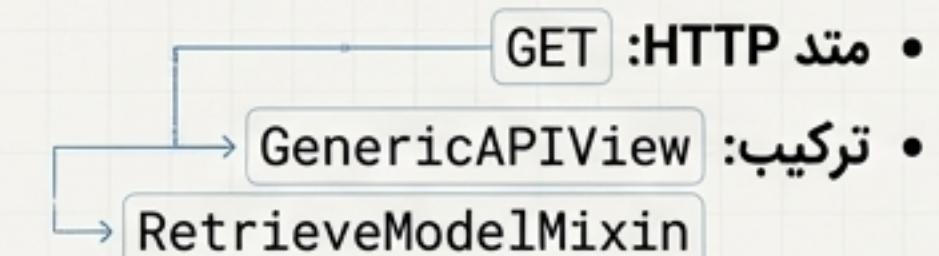
ListAPIView

- کاربرد: اندپوینت‌های فقط-خواندنی (only) برای نمایش مجموعه‌ای از نمونه‌ها.



RetrieveAPIView

- کاربرد: اندپوینت‌های فقط-خواندنی (only) برای نمایش یک نمونه‌ی واحد.



DestroyAPIView

- کاربرد: فقط برای اندپوینت‌های حذف (delete-only) برای یک نمونه‌ی واحد.



UpdateAPIView

- کاربرد: فقط برای اندپوینت‌های آپدیت (update-only) برای یک نمونه‌ی واحد.



مرجع کلاس‌های آماده (بخش دوم: عملیات ترکیبی)

این کلاس‌ها چندین عملیات را با هم ترکیب کرده و الگوهای رایج API را پوشش می‌دهند.

ListCreateAPIView

- کاربرد: اندپوینت‌های خواندن-نوشتن (read-write) برای مجموعه‌ای از نمونه‌ها.
- متدهای HTTP : GET, POST
- ترکیب: GenericAPIView, ListModelMixin, CreateModelMixin



RetrieveUpdateAPIView

- کاربرد: اندپوینت‌های خواندن یا آپدیت (read or update) برای یک نمونه‌ی واحد.
- متدهای HTTP : GET, PUT, PATCH
- ترکیب: GenericAPIView, RetrieveModelMixin, UpdateModelMixin



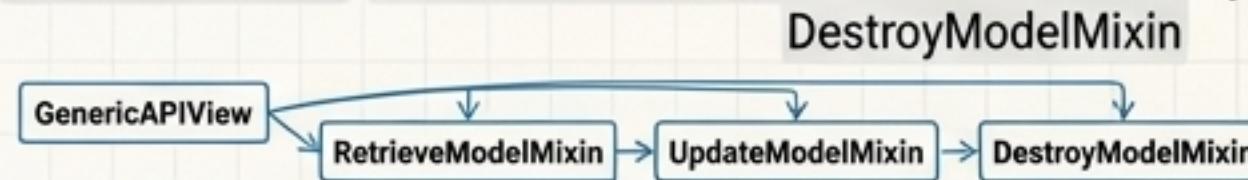
RetrieveDestroyAPIView

- کاربرد: اندپوینت‌های خواندن یا حذف (read or delete) برای یک نمونه‌ی واحد.
- متدهای HTTP : GET, DELETE
- ترکیب: GenericAPIView, RetrieveModelMixin, DestroyModelMixin



RetrieveUpdateDestroyAPIView

- کاربرد: اندپوینت‌های خواندن-نوشتن-حذف (read-write-delete) برای یک نمونه‌ی واحد.
- متدهای HTTP : GET, PUT, PATCH, DELETE
- ترکیب: GenericAPIView, RetrieveModelMixin, UpdateModelMixin, DestroyModelMixin



بخش چهارم: سفارشی‌سازی و نکات تکمیلی

بلوپرینت‌های پیشرفته: سفارشی‌سازی ها Generic View ها

وقتی رفتار پیش‌فرض کافی نیست، می‌توانید با ساختن اجزای خود، فریم‌ورک را توسعه دهید.

استراتژی ۲: ساخت کلاس‌های پایه‌ی سفارشی (Custom Base Classes)

- چه زمانی؟ وقتی یک رفتار سفارشی را به طور مداوم در تعداد زیادی از view‌های پروژه تکرار می‌کنید.

```
class MultipleFieldLookupMixin:  
    """  
    Apply this mixin to get multiple field filtering.  
    """  
  
    def get_object(self):  
        queryset = self.get_queryset()  
        queryset = self.filter_queryset(queryset)  
        filter = {}  
        for field in self.lookup_fields:  
            if self.kwargs.get(field):  
                filter[field] = self.kwargs[field]  
        obj = get_object_or_404(queryset, **filter)  
        self.check_object_permissions(self.request, obj)  
        return obj  
  
class RetrieveUserView(MultipleFieldLookupMixin,  
                      generics.RetrieveAPIView):  
    queryset = User.objects.all()  
    serializer_class = UserSerializer  
    lookup_fields = ['account', 'username']
```

استراتژی ۱: ساخت میکسین‌های سفارشی (Custom Mixins)

- چه زمانی؟ وقتی یک رفتار خاص و قابل استفاده‌ی مجدد دارد که می‌خواهد view‌های مختلف اضافه کنید.

```
class MultipleFieldLookupMixin:  
    """  
    Apply this mixin to get multiple field filtering.  
    """  
  
    def get_object(self):  
        queryset = self.get_queryset()  
        queryset = self.filter_queryset(queryset)  
        filter = {}  
        for field in self.lookup_fields:  
            if self.kwargs.get(field):  
                filter[field] = self.kwargs[field]  
        obj = get_object_or_404(queryset, **filter)  
        self.check_object_permissions(self.request, obj)  
        return obj  
  
class RetrieveUserView(MultipleFieldLookupMixin,  
                      generics.RetrieveAPIView):  
    queryset = User.objects.all()  
    serializer_class = UserSerializer  
    lookup_fields = ['account', 'username']
```

ملاحظات مهم و بسته‌های جانبی

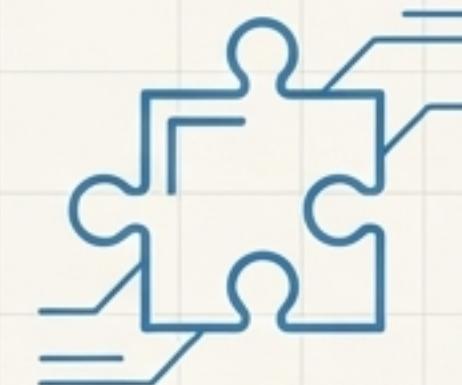
بخش اول: تاریخچه‌ی رفتار `PUT as create`

توضیح: قبل از نسخه‌ی ۳.۰، DRF درخواست‌های `PUT` را اگر آبجکت وجود نداشت، به عنوان یک عملیات "ایجاد" در نظر می‌گرفت.



بخش دوم: بسته‌های شخص ثالث (Third Party Packages)

مقدمه: اکوسیستم DRF فراتر از قابلیت‌های داخلی آن است.



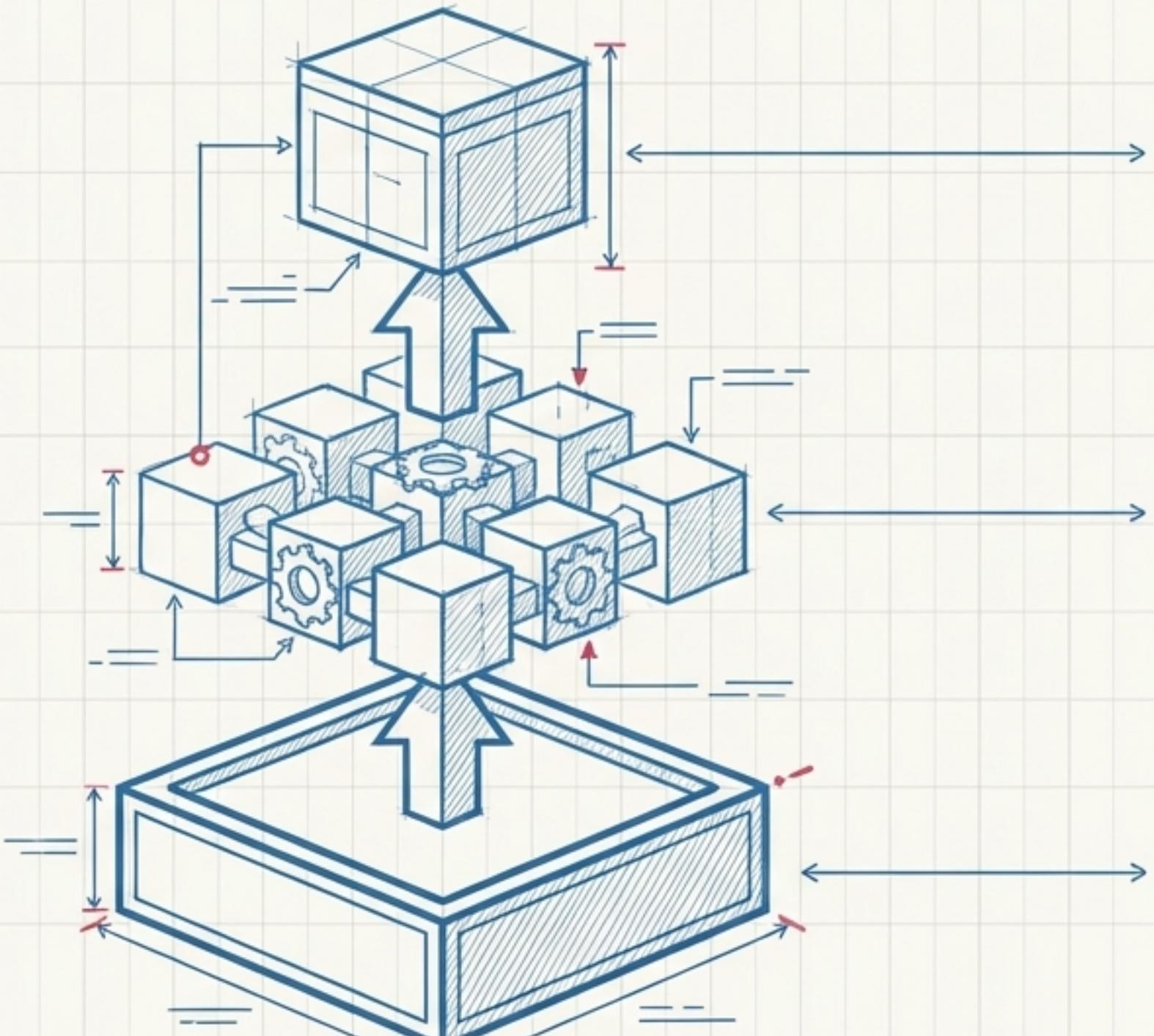
`Django Rest Multiple Models`

کاربرد بسته: این بسته یک view و mixin برای ارسال چندین مدل و/یا کوئری‌ست سریالایز شده در یک درخواست API واحد فراهم می‌کند.

```
pip install django-rest-multiple-models
```

جمع‌بندی: معماری Generic View در یک نگاه

تسلط بر Generic View ها، تسلط بر درگ نحوه‌ی تعامل سه بخش اصلی آن است.



1. شالوده (The Foundation)
نقش: فراهم کردن وضعیت (State) و زمینه (Context).
.serializer و queryset مانند

2. قطعات (The Components)
نقش: فراهم کردن عملیات (Actions) و رفتار (Behavior). مانند (.create() و .list())

3. محصول (The Result)
نقش: ترکیب شالوده و قطعات برای ساختن اندیپوینت‌های آماده‌ی مصرف.

با درگ این معماری، شما نه تنها می‌توانید از view های آماده به طور مؤثر استفاده کنید، بلکه می‌توانید فریم‌ورک را برای حل هر نیازی توسعه دهید.