

# Django REST API بهینه: تسلط کامل بر ViewSet‌ها

راهنمای جامع برای نوشتن کدهای تمیزتر، سریع‌تر و مقیاس‌پذیرتر

# نقطه شروع: منطق تکراری در View‌های سنتی

```
# views.py

class UserList(APIView):
    def get(self, request, format=None):
        queryset = User.objects.all() #<--  
        serializer = UserSerializer(queryset,
many=True)
        return Response(serializer.data)
```

```
# views.py

class UserDetail(APIView):
    def get(self, request, pk, format=None):
        queryset = User.objects.all() #<--  
        user = get_object_or_404(queryset, pk=pk)
        serializer = UserSerializer(user)
        return Response(serializer.data)
```

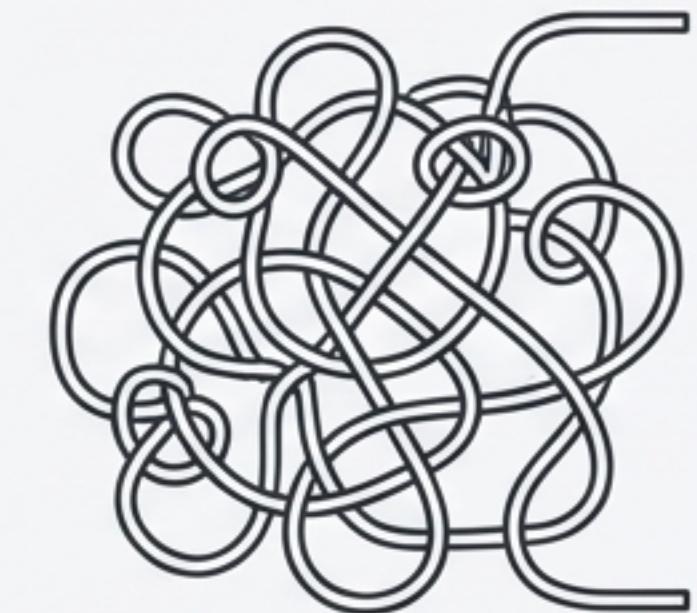
چالش اصلی: `queryset` در چندین کلاس تکرار شده است. این یعنی نقض اصل DRY (Don't Repeat Yourself) است. و افزایش احتمال خطا هنگام تغییرات.

# چالش دوم: مدیریت دستی و پیچیدهی URLها

```
# urls.py
from django.urls import path
from myapp.views import UserList, UserDetail

urlpatterns = [
    path('users/', UserList.as_view()),
    path('users/<int:pk>', UserDetail.as_view()),
    # ...
]
```

تصور کنید برای هم مسای جدآگانهای تعریف شود



با افزایش تعداد مدل‌ها و متدهای URLconf، فایل HTTP به سرعت شلوغ، مدیریت آن دشوار و مستعد خطا می‌شود. این روش مقیاس‌پذیر نیست.

# راه حل: ViewSet - تجمعی منطق و معرفی اکشن‌ها

یک نوعی View مبتنی بر کلاس است که به جای متدهایی مثل `get` یا `post`، اکشن‌بایی (actions) مانند `list` و `create` را ارائه می‌دهد. این اکشن‌ها در زمان نهایی‌سازی View با متدهای `as_view()`.. به متدهای HTTP متصل می‌شوند.

```
from rest_framework import viewsets
from rest_framework.response import Response

class UserViewSet(viewsets.ViewSet):
    """
    یک ViewSet ساده برای نمایش لیست یا یک کاربر خاص.
    """

    def list(self, request):
        queryset = User.objects.all()
        serializer = UserSerializer(queryset, many=True)
        return Response(serializer.data)

    def retrieve(self, request, pk=None):
        queryset = User.objects.all()
        user = get_object_or_404(queryset, pk=pk)
        serializer = UserSerializer(user)
        return Response(serializer.data)
```



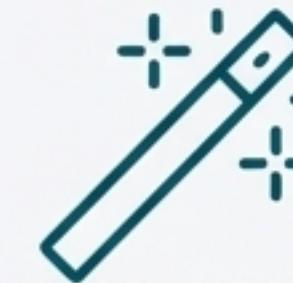
# قدرت واقعی: سیمکشی خودکار URL‌ها با Router

## روش سنتی



```
# urls.py
urlpatterns = [
    path('users/', UserList.as_view()),
    path('users/<int:pk>', UserDetail.as_view()),
    # ... و بیشتر
]
```

## ViewSet روش



```
from myapp.views import UserViewSet
from rest_framework.routers import DefaultRouter

router = DefaultRouter()
router.register(r'users', UserViewSet, basename='user')

urlpatterns = router.urls
```

روتربهای طور خودکار URLconf را برای شما تولید می‌کنند. دیگر نیازی به تعریف دستی هر مسیر نیست. این کار باعث ایجاد یک ساختار URL منسجم در کل API شما می‌شود.

# جعبه ابزار کلاس پایه مناسب: ViewSet



## 'ViewSet'

پایه‌ای‌ترین کلاس. از 'APIView' ارث بری می‌کند. هیچ افع به صورت پیش‌فرض پیاده‌سازی نشده است. برای زمانی که به کنترل کامل بر روی منطق نیاز دارید.



## 'GenericViewSet'

از 'GenericAPIView' ارث بری می‌کند. متدهای کمکی مانند 'get\_object' و 'get\_queryset' فراهم می‌کند، اما همچنان هیچ اکشنی ندارد. برای ترکیب با Mixin‌ها ایده‌آل است.



## 'ReadOnlyModelViewSet'

اکشن‌های فقط-خواندنی ('list' و 'retrieve') را به صورت آماده فراهم می‌کند. مناسب برای نقاط پایانی که فقط وظیفه نمایش داده‌ها را دارند.



## 'ModelViewSet'

نیروگاه کامل. تمام اکشن‌های ('list', 'create', 'retrieve', 'update', 'partial\_update', 'destroy') CRUDe را با استفاده از 'auto' Mixin‌ها پیاده‌سازی می‌کند. سریع‌ترین راه برای ساخت یک API کامل برای یک مدل.

# اوج بھرھوری: یک API کامل با `ModelViewSet`

```
class UserViewSet(viewsets.ModelViewSet):
```

```
    """
```

یک ViewSet برای مشاهده و ویرایش نمونه‌های کاربر.

```
    """
```

```
    queryset = User.objects.all()
```

```
    serializer_class = UserSerializer
```



## ۱. منطق مرکزی:

فقط یک بار تعریف می‌شود و در تمام اکشن‌ها استفاده می‌شود.



## ۲. عدم نیاز به تنظیم URL:

روت‌ها نیاز به تنظیم URL: روت‌ها بقیه کار را انجام می‌دهند.

# دیکشنری اکشن‌ها: نگاشت اکشن‌های استاندارد به URL

| اکشن (Action)  | متدهای HTTP و الگوی URL Pattern |
|----------------|---------------------------------|
| list           | GET /users/                     |
| create         | POST /users/                    |
| retrieve       | GET /users/{pk}/                |
| update         | PUT /users/{pk}/                |
| partial_update | PATCH /users/{pk}/              |
| destroy        | DELETE /users/{pk}/             |

این مسیرها به صورت پیش‌فرض توسط `DefaultRouter` تولید می‌شوند.

# فراتر از CRUD: افزودن نقاط پایانی سفارشی با `@action`

گاهی اوقات شما به نقاط پایانی نیاز دارید که در عملیات استاندارد CRUD قرار نمی‌گیرند. دکوراتور `@action` به شما اجازه می‌دهد متدهای دلخواه را برای روتینگ علامت‌گذاری کنید.

```
from rest_framework.decorators import action

class UserViewSet(viewsets.ModelViewSet):
    queryset = User.objects.all()
    serializer_class = UserSerializer

    @action(detail=True, methods=['post']) #<-->
    def set_password(self, request, pk=None):
        user = self.get_object()
        # ... (implementation logic)
        return Response({'status': 'password set'})
```



## `detail=True`

این اکشن بر روی یک نمونه خاص از مدل (یک object) عمل می‌کند. URL شامل `pk` خواهد بود `(/users/{pk}/set_password/)`.



## `detail=False`

این اکشن بر روی کل مجموعه (collection) عمل می‌کند. URL شامل `pk` نخواهد بود `(/users/recent_users/)`.

# کنترل دقیق: سفارشی‌سازی رفتار اکشن‌ها

- تعیین متد های HTTP مجاز

```
from rest_framework.decorators import action
class UserViewSet(viewsets.ModelViewSet):
    ...
    @action(detail=True, methods=['post', 'delete']) #<-- `methods` parameter highlighted
    def unset_password(self, request, pk=None): ...
```

اجازه دادن به متد های خاص HTTP (مانند POST و DELETE) برای یک اکشن.

- بازنویسی تنظیمات ViewSet (مانند دسترسی‌ها)

```
from rest_framework.decorators import action
from rest_framework.permissions import IsAdminUser
class UserViewSet(viewsets.ModelViewSet):
    ...
    @action(detail=True, methods=['post'], permission_classes=[IsAdminUser]) #<-- `permission_classes` highlighted
    def set_password(self, request, pk=None): ...
```

می‌توانید تنظیمات ViewSet مانند `permission\_classes` را برای یک اکشن خاص بازنویسی کنید.

- تغییر نام URL و مسیر

```
from rest_framework.decorators import action
class UserViewSet(viewsets.ModelViewSet):
    ...
    @action(detail=True, url_path='change-password', url_name='change_password') #<-- url parameters highlighted
    def set_password(self, request, pk=None): ...
```

\*\*نکته\*\*: URL `users/{pk}/change-password` را به `users/{pk}/change-password` تغییر می‌کند.

# منطق پویا: تصمیم‌گیری بر اساس اکشن فعلی

درون یک متده ViewSet، شما می‌توانید به صفات (attributes) خاصی دسترسی پیدا کنید تا رفتار را بر اساس درخواست فعلی تنظیم کنید.

## صفات کلیدی (Key Attributes)

'list': نام اکشن فعلی به صورت رشته (مثلًا 'self.action'  یا 'create' یا 'detail' یا نام اکشن سفارشی شما).

'detail': یک مقدار بولین که نشان می‌دهد اکشن برای detail است ('True') یا list ('False'). 

## نمونه کد عملی (Practical Code Example)

```
def get_permissions(self):
    """
    سطوح دسترسی را بر اساس اکشن فعلی به صورت پویا تعیین می‌کند.
    """
    if self.action == 'list':
        # برای لیست کاربران، فقط احراز هویت کافی است
        permission_classes = [IsAuthenticated]
    else:
        # برای سایر عملیات، کاربر باید ادمین باشد
        permission_classes = [IsAdminUser]
    return [permission() for permission in permission_classes]
```

# مهندسى معکوس URL: استفاده صحیح از `reverse\_action`

## چرا مهم است؟ (Why it Matters?)

برای به دست آوردن URL یک اکشن (چه استاندارد و چه سفارشی) به صورت برنامه‌نویسی، هرگز نباید URL را به صورت دستی (hard-code) بسازید. این کار باعث شکنندگی کد شما می‌شود.

## راه حل (The Solution)

متدهای `.reverse_action()`

## نحوه استفاده (How to Use)

# فرض کنید 'view' یک نمونه از ViewSet ما است  
# و request در آن موجود است.

```
url = view.reverse_action("set-password", args=["1"]) # معکوس کردن URL برای یک اکشن سفارشی  
# نتیجه 'http://.../api/users/1/set_password/'  
  
# همچنین می‌توانید از url_name تعریف شده در دکوراتور استفاده کنید  
url = view.reverse_action(view.set_password.url_name, args=["1"])
```

## نکته مهم (Important Note)

این متدهای به صورت خودکار از `basename` که توسط روت تعريف شده استفاده نمی‌کنند. اگر از روت تعريف شده استفاده نمی‌کنید، باید `basename` را در متدهای `.as_view()` مشخص کنید.

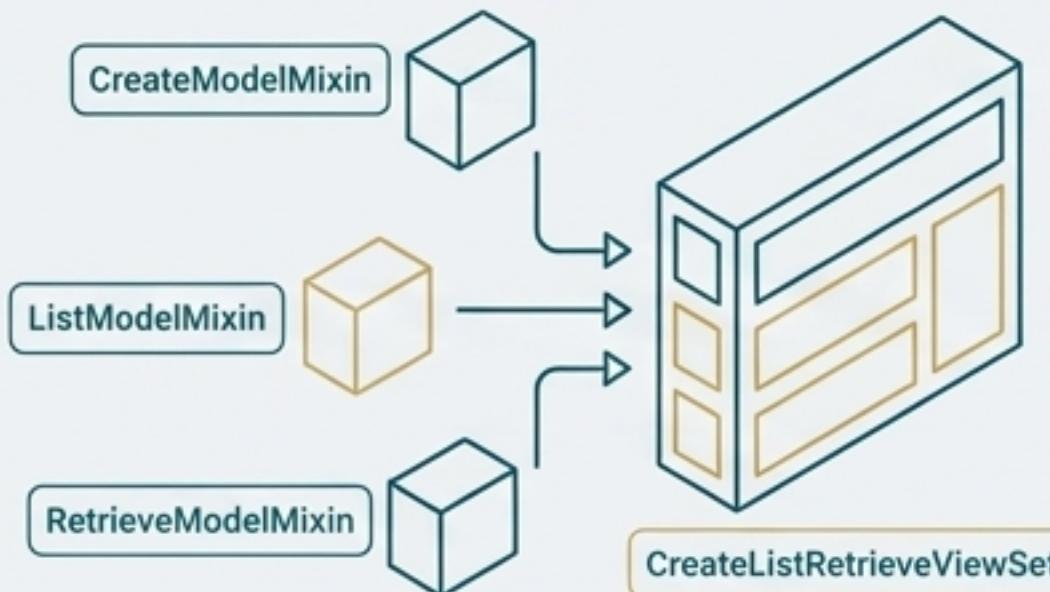
# معماری پیشرفته: ساخت ViewSet با Mixin ها

## هدف (Goal)

برای الگوهای رایج در API خود، به جای تکرار کد، می‌توانید کلاس‌های پایه قابل استفاده مجدد بسازید.

## روش (Method)

با ارث بری از `Mixin` و `GenericViewSet` های مورد نیاز، `ViewSet` های سفارشی خود را بسازید.



```
from rest_framework import mixins, viewsets

class CreateListRetrieveViewSet(mixins.CreateModelMixin,
                                mixins.ListModelMixin,
                                mixins.RetrieveModelMixin,
                                viewsets.GenericViewSet):
    """
    """

    pass
```

یک `ViewSet` که اکشن‌های `list` و `retrieve` را فراهم می‌کند. برای استفاده، کافی است از آن ارث بری کرده و صفات `serializer\_class` و `queryset` را تنظیم کنید.

# یک نمونه کامل: ترکیب `ModelViewSet` و اکشن‌های سفارشی

در این مثال، ما از قدرت `ModelViewSet` برای اکشن‌های استاندارد CRUD استفاده می‌کنیم و همزمان دو اکشن سفارشی برای نیازهای خاص خود اضافه می‌کنیم.

```
class UserViewSet(viewsets.ModelViewSet):
```

```
    ....
```

یک ViewSet که هم اکشن‌های استاندارد و هم سفارشی را ارائه می‌دهد.

```
....
```

```
queryset = User.objects.all()
serializer_class = UserSerializer
```

اکشن برای یک آبجکت خاص #

```
@action(detail=True, methods=['post'])
```

```
def set_password(self, request, pk=None):
```

# ... (implementation) ...

اکشن برای کل مجموعه #

```
@action(detail=False)
```

```
def recent_users(self, request):
```

# ... (implementation) ...



اکشن **Detail**: روی یک کاربر خاص (`/users/{pk}/...`) عمل می‌کند.



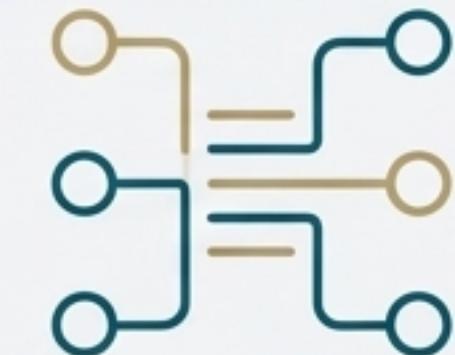
اکشن **List**: روی کل مجموعه (`/users/...`) عمل می‌کند.

# جمع‌بندی: چرا ViewSet‌ها یک تغییردهنده بازی هستند؟

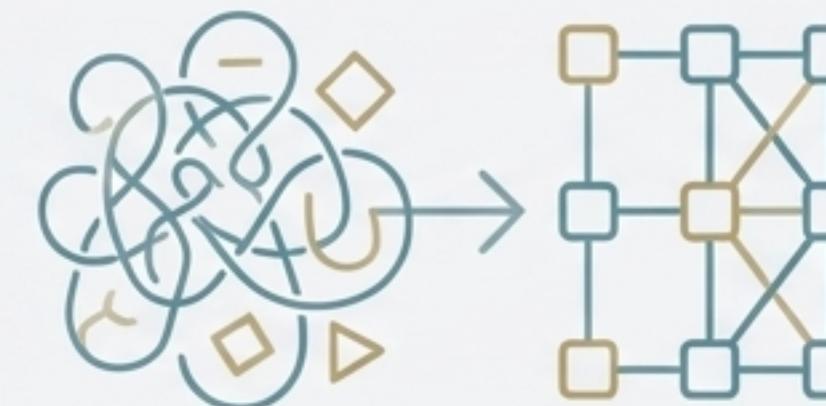


۱. کد کمتر، منطق متمرکز (DRY): با تجمع منطق مرتبط در یک کلاس، از تکرار کد جلوگیری می‌کنید.

۲. پیکربندی URL منسجم و خودکار: روت‌ها ثبات و سادگی را در سراسر API شما تضمین می‌کنند.



۳. توسعه سریع API: با ModelViewSet می‌توانید نقاط پایانی کامل CRUD را در عرض چند دقیقه بسازید.



۴. انعطاف‌پذیری و قدرت بالا: با اکشن‌های سفارشی، منطق پویا و کلاس‌های پایه، کنترل کامل برای پیاده‌سازی هر نیازی در دستان شماست.

