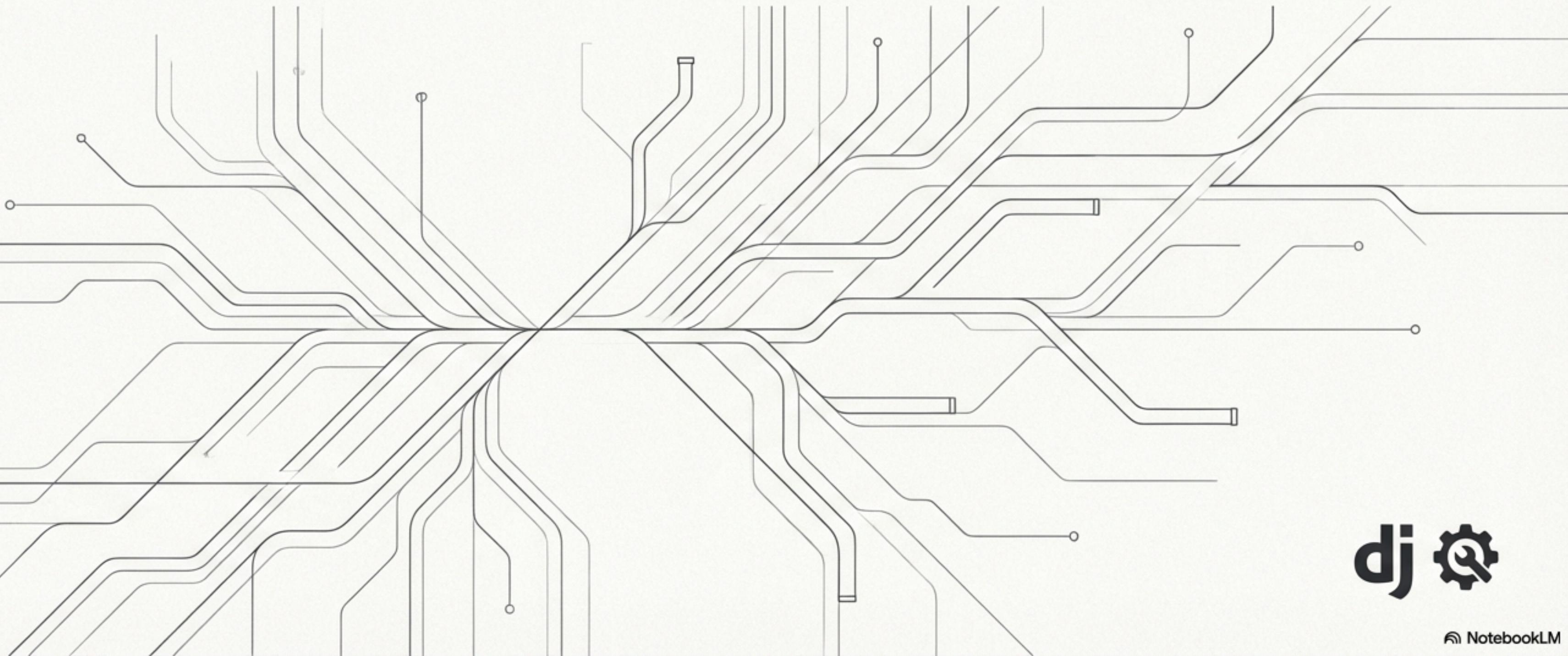


تسلط بر Router ها در Django REST Framework

راهنمای جامع برای تعریف خودکار URLها، از اصول اولیه تا سفارشی‌سازی کامل



چرا Router در تعريف URL حذف تکرار در تعريف

بعضی از فریمورک‌های وب مانند Rails، قابلیتی برای تعیین خودکار نگاشت URL‌ها به منطق برنامه ارائه می‌دهند. این قابلیت را به جنگو اضافه می‌کند و روشی ساده، سریع و سازگار برای اتصال منطق View به مجموعه‌ای از URL‌ها فراهم می‌آورد.

رویکرد دستی: تکراری و مستعد خطا

```
from django.urls import path
from .views import UserViewSet

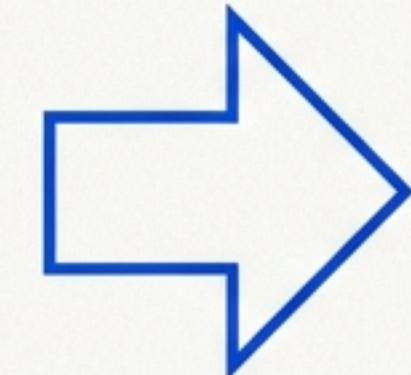
user_list = UserViewSet.as_view({'get': 'list',
                                'post': 'create'})
user_detail = UserViewSet.as_view({'get': 'retrieve',
                                    'put': 'update',
                                    'patch': 'partial_update',
                                    'delete': 'destroy'})

urlpatterns = [
    path('users/', user_list, name='user-list'),
    path('users/<int:pk>/', user_detail,
         name='user-detail'),
]
```

رویکرد خودکار با Router: سریع و سازگار

```
from rest_framework import routers
router = routers.SimpleRouter()
router.register(r'users', UserViewSet)

urlpatterns = router.urls
```



ابزار اصلی: آشنایی با `SimpleRouter`

```
from rest_framework import routers  
  
router = routers.SimpleRouter()  
  
router.register(r'users', UserViewSet)  
router.register(r'accounts', AccountViewSet)  
  
urlpatterns = router.urls
```

سلاجه register() method

prefix پیشوند URL که برای این مجموعه از route‌ها استفاده می‌شود.

viewset .ViewSet کلاس

basename (اختیاری): پایه‌ای که برای نامگذاری URL‌های تولید شده استفاده می‌شود. اگر مشخص نشود، basename، queryset به طور خودکار بر اساس اtribیوت در viewset تولید می‌شود.

خروجی `SimpleRouter`: الگوهای URL تولید شده

کد اسلاید قبل، الگوهای URL زیر را تولید می‌کند:

URL Pattern)	HTTP متدها	اکشن (Action)	نام (URL Name)
prefix='users'	^users/\$	GET POST	user-list ————— prefix='users'
	^users/{pk}/\$	GET PUT PATCH DELETE	user-detail
prefix='accounts'	^accounts/\$	GET POST	account-list ————— prefix='accounts'
	^accounts/{pk}/\$	GET PUT PATCH DELETE	account-detail

نکته کلیدی: چه زمانی `basename` است؟

معمولًا نیازی به تعیین `basename` نیست. اما اگر ViewSet شما اتربیوت ".queryset" را نداشته باشد (مثلاً متدهای `get_queryset` را بازنویسی کرده باشید)، باید `basename` را به صراحة مشخص کنید.



مشکل

'basename' argument not specified, and could not automatically determine the name from the viewset, as it does not have a '.queryset' attribute.



راه حل

```
# ViewSet without a .queryset attribute
class ReportViewSet(viewsets.ViewSet):
    def get_queryset(self):
        return Reports.objects.all()

# Registration requires basename
router.register(r'reports', ReportViewSet, basename='report')
```

ادغام `urlpatterns` با Router پروژه

۱. الحق مستقیم (Direct Append)

```
urlpatterns += router.urls
```

۲. استفاده از `include`

```
from django.urls import  
path, include  
  
urlpatterns = [  
    path('api/',  
        include(router.urls)),  
]
```

۳. استفاده از `include` با Namespace

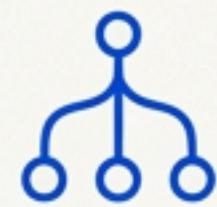
```
urlpatterns = [  
    path('api/',  
        include((router.urls,  
        'app_name'))),  
]
```

نکته مهم*: اگر از namespace با سریالایزرهای هایپرلینک شده استفاده می‌کنید، باید پارامتر `view_name` در فیلدۀای سریالایزر را متناسب با آن تنظیم کنید. مثال: `.view_name='app_name:user-detail'.`



یک پله بالاتر: `DefaultRouter` و امکانات بیشتر آن

است اما دو قابلیت کلیدی اضافه دارد:



1. **نمای ریشه API (API Root View)**: یک نقطه پایانی (endpoint) اصلی ایجاد می‌کند که لیستی از هایپرلینک‌ها به تمام view‌های لیست را نمایش می‌دهد.



2. **پشتیبانی از پسوندهای فرمت**: به طور خودکار مسیرهایی برای پسوندهای `json` و امثال آن ایجاد می‌کند.

SimpleRouter

^users/\$
^users/{pk}/\$



DefaultRouter

^\$ (API Root)
^users/\$
^users\.(?P<format>[a-zA-Z0-9]+)\/\$ (Format Suffix)
^users/{pk}/\$
^users/{pk}\.(?P<format>[a-zA-Z0-9]+)\/\$ (Format Suffix)

فراتر از استانداردها: افزودن اکشن‌های سفارشی با `@action`

یک ViewSet می‌تواند با استفاده از دکوراتور `@action`، اکشن‌های اضافی را برای مسیریابی مشخص کند. این اکشن‌های اضافی در مسیرهای تولید شده گنجانده خواهند شد.

```
from rest_framework.decorators import action

class UserViewSet(ModelViewSet):
    # ... existing code ...

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

Output

`^users/{pk}/set_password/$`

`'user-set-password'`

الگوی URL تولید شده:

نام URL تولید شده:

سفارشی سازی مسیر و نام اکشن ها

به طور پیش فرض، الگوی URL بر اساس نام متده و نام URL از ترکیب `basename` و نام متده ساخته می شود. برای تغییر این رفتار، از آرگومان های `url_name` و `url_path` در دکوراتور @action استفاده کنید.

پیش فرض

```
@action(methods=['post'], detail=True)
def set_password(self, request, pk=None): ...
# URL: ^users/{pk}/set_password/$
# Name: 'user-set-password'
```

سفارشی

```
@action(methods=['post'], detail=True,
        url_path='change-password',
        url_name='change_password')
def set_password(self, request, pk=None): ...
# URL: ^users/{pk}/change-password/$
# Name: 'user-change_password'
```

گزینه‌های پیشرفته: تنظیمات دقیق مسیریابی

کنترل Slash

به طور پیش‌فرض، URLs با یک اسلش (/) در انتهای ساخته می‌شوند. برای غیرفعال کردن این رفتار، از `trailing_slash=False` هنگام ساخت روت استفاده کنید.

```
router = SimpleRouter(trailing_slash=False)
```

/users/ → /users

استفاده از `path()` جنگو

برای استفاده از `path()` و `path converters` جنگو به جای `re_path()` (که پیش‌فرض است)، از `use_regex_path=False` استفاده کنید.

```
router = SimpleRouter(use_regex_path=False)
```

re_path() → path()

سفارشی‌سازی Lookup Field

برای تطبیق الگوهای خاص در URL (مانند UUID)، می‌توانید اتریبیوت `lookup_value_pk` یا `lookup_value_regex` ViewSet را روی تنظیم کنید.

```
class MyViewSet(ViewSet):  
    lookup_value_converter = 'uuid'
```

/{pk}/ → /{uuid:pk}/

کنترل کامل: ساخت یک Router سفارشی

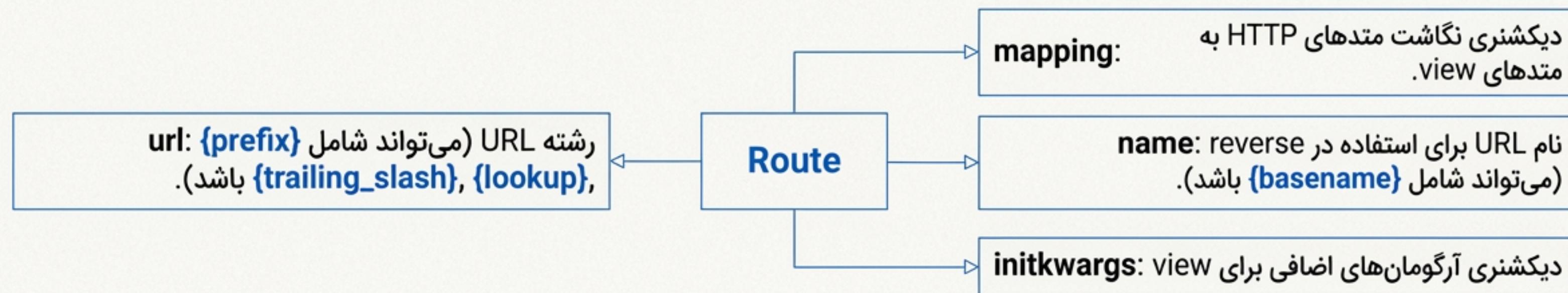
چرا یک Router سفارشی؟

پیاده‌سازی یک روتر سفارشی به ندرت لازم است، اما زمانی مفید است که شما نیازمندی‌های خاصی در مورد ساختار URL‌های API خود دارید.

مفهوم اصلی

ساده‌ترین راه، ارث بری از یکی از کلاس‌های روتر موجود و بازنویسی اtribیوت `routes`. است. این اtribیوت لیستی از `DynamicRoute` و `Route` و `namedtuple` است.

بلوک‌های سازنده: `Route`



مثال عملی: پیاده‌سازی `CustomReadOnlyRouter`

Blueprint

```
from rest_framework.routers import Route, SimpleRouter
class CustomReadOnlyRouter(SimpleRouter):
    """
    A router for read-only APIs, which doesn't use trailing slashes.
    """
    routes = [
        Route(
            url=r'^{prefix}$',
            mapping={'get': 'list'},
            name='{basename}-list',
            detail=False,
            initkwargs={'suffix': 'List'}
        ),
        Route(
            url=r'^{prefix}/{lookup}$',
            mapping={'get': 'retrieve'},
            name='{basename}-detail',
            detail=True,
            initkwargs={'suffix': 'Detail'}
        )
    ]
```

```
router = CustomReadOnlyRouter()
router.register('users', UserViewSet, basename='user')
```

استفاده و خروجی

Final Structure

URL	متد	اکشن	نام
/users	GET	list	user-list
/users/{username}	GET	retrieve	user-detail

فراتر از هسته: پکیج‌های شخص ثالث (Third Party)

برای نیازهای پیچیده‌تر، همیشه لازم نیست از صفر شروع کنید. پکیج‌های زیر قابلیت‌های بیشتری را ارائه می‌دهند:



DRF Nested Routers

برای کار با منابع تودرتو (resources) روتهای و فیلدات ارتباطی فراهم (nested) فراهم می‌کند.



ModelRouter (wq.db.rest)

یک کلاس `ModelRouter` پیشرفته ارائه می‌دهد که با یک API شبیه به `admin.site.register` کار می‌کند. کافیست مدل را ثبت کنید تا URL، سرلایزر و viewset به صورت خودکار استنتاج شوند.



DRF-extensions

روتهایی برای ساخت viewsets تودرتو و کنترلهای سطح collection با نام‌های قابل تنظیم ارائه می‌دهد.

جمع‌بندی: نقشه راه استفاده از Routerها

1 انتخاب پیش‌فرض (برای ۹۵٪ موارد)

از `DefaultRouter` و `SimpleRouter` استفاده کنید. آنها سریع، قابل اعتماد و پاسخگوی اکثر نیازها هستند.



2

گسترش قابلیت‌ها (برای مسیرهای خاص)

با دکوراتور `@action` قابلیت‌های ViewSet خود را برای مسیرهای سفارشی گسترش دهید.



3

کنترل کامل (برای ساختارهای منحصر به فرد)

با ساخت یک Router سفارشی، ساختار URL مورد نظر خود را به صورت کامل و قابل استفاده مجدد تعریف کنید.



4

سناریوهای پیچیده (برای نیازهای پیشرفته)

برای الگوهای پیچیده مانند منابع تودرتو، اکوسیستم پکیج‌های شخص ثالث را بررسی کنید.

