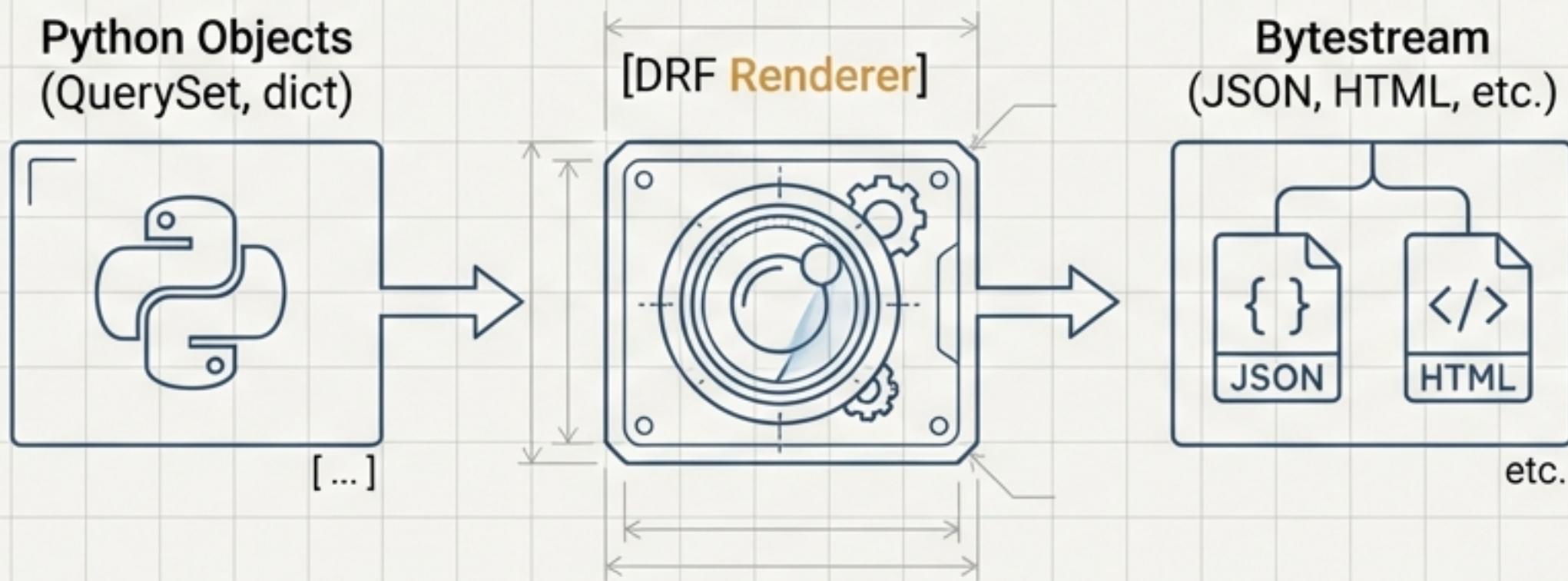


# راهنمای جامع در Django REST Framework

از مفاهیم پایه تا پیادهسازی‌های پیشرفته

# رندرینگ (Rendering) : ترجمه‌ی داده‌های پایتون برای کلاینت



ها در DRF به عنوان پلی بین داده‌های داخلی پایتون (مانند QuerySet و دیکشنری‌ها) و جریان نهایی بایت‌ها (bytestream) که به کلاینت ارسال می‌شود (مانند JSON، ارسال و HTML، XML، فرآنا، HTML و غیره) عمل می‌کنند. آنها آخرین مرحله قبل از ارسال پاسخ هستند.

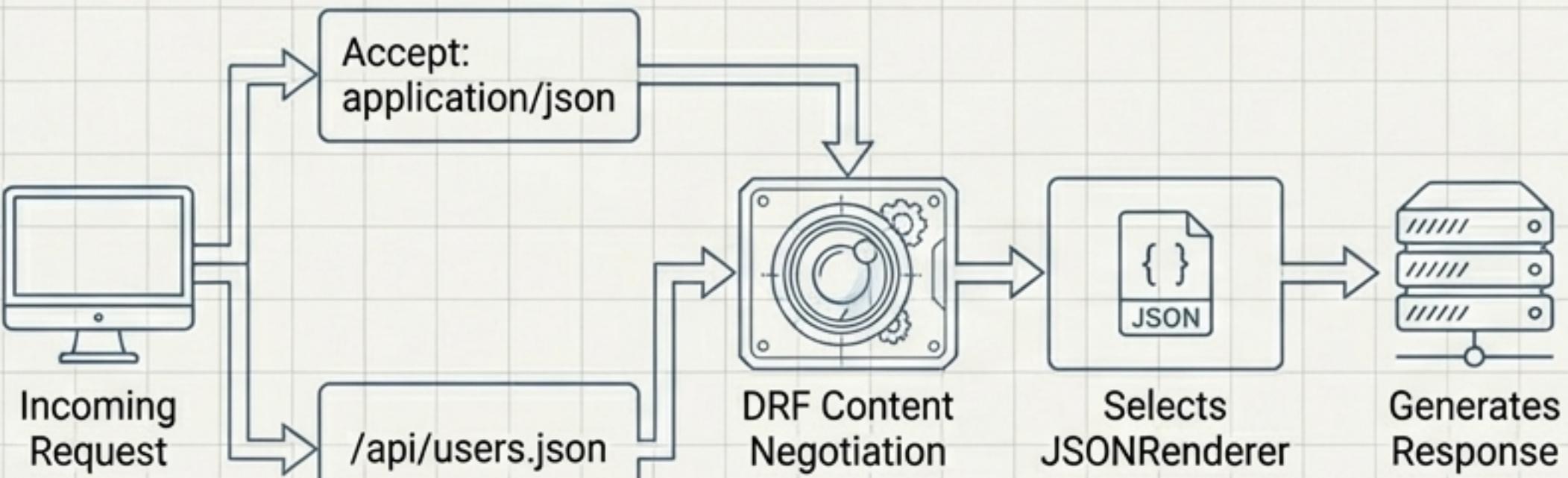
Before a TemplateResponse instance can be returned to the client, it must be rendered...  
The rendering process takes the intermediate representation of template and context, and turns it into the final byte stream that can be served to the client."

# Content Negotiation می‌کند؟ فرآیند را انتخاب چگونه Renderer DRF

## دو مکانیسم اصلی

۱. هدر **Accept**: فرمت ترجیحی کلاینت (مثال: `(Accept: application/json)`)
۲. پسوندهای فرمت :**(Format Suffixes)** درخواست صریح فرمت در `/api/users.json` (مثال).

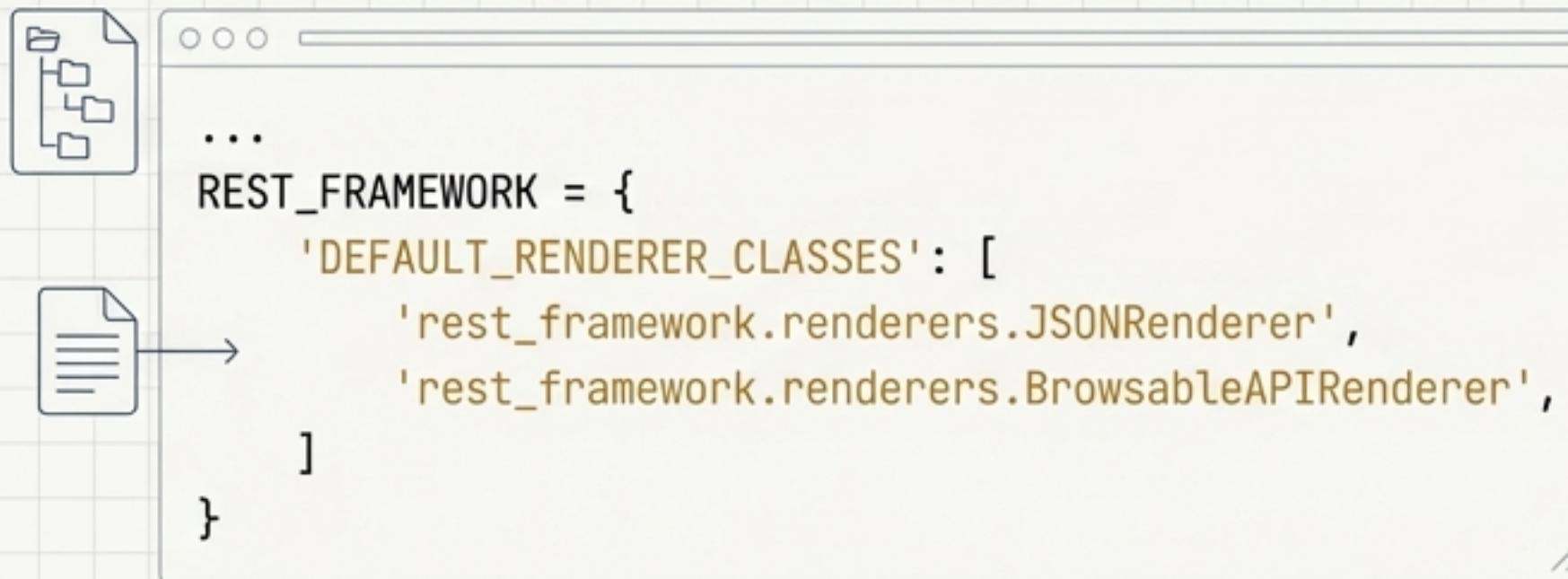
**نکته کلیدی:** DRF بر اساس درخواست، مناسب‌ترین رندر را انتخاب می‌کند.



# پیکربندی Rendererها: در سطح پروژه و در سطح View

## پیکربندی سراسری (settings.py در در

این تنظیمات، رندرهای پیشفرض را برای کل پروژه تعیین می‌کند.



```
...  
REST_FRAMEWORK = {  
    'DEFAULT_RENDERER_CLASSES': [  
        'rest_framework.renderers.JSONRenderer',  
        'rest_framework.renderers.BrowsableAPIRenderer',  
    ]  
}
```

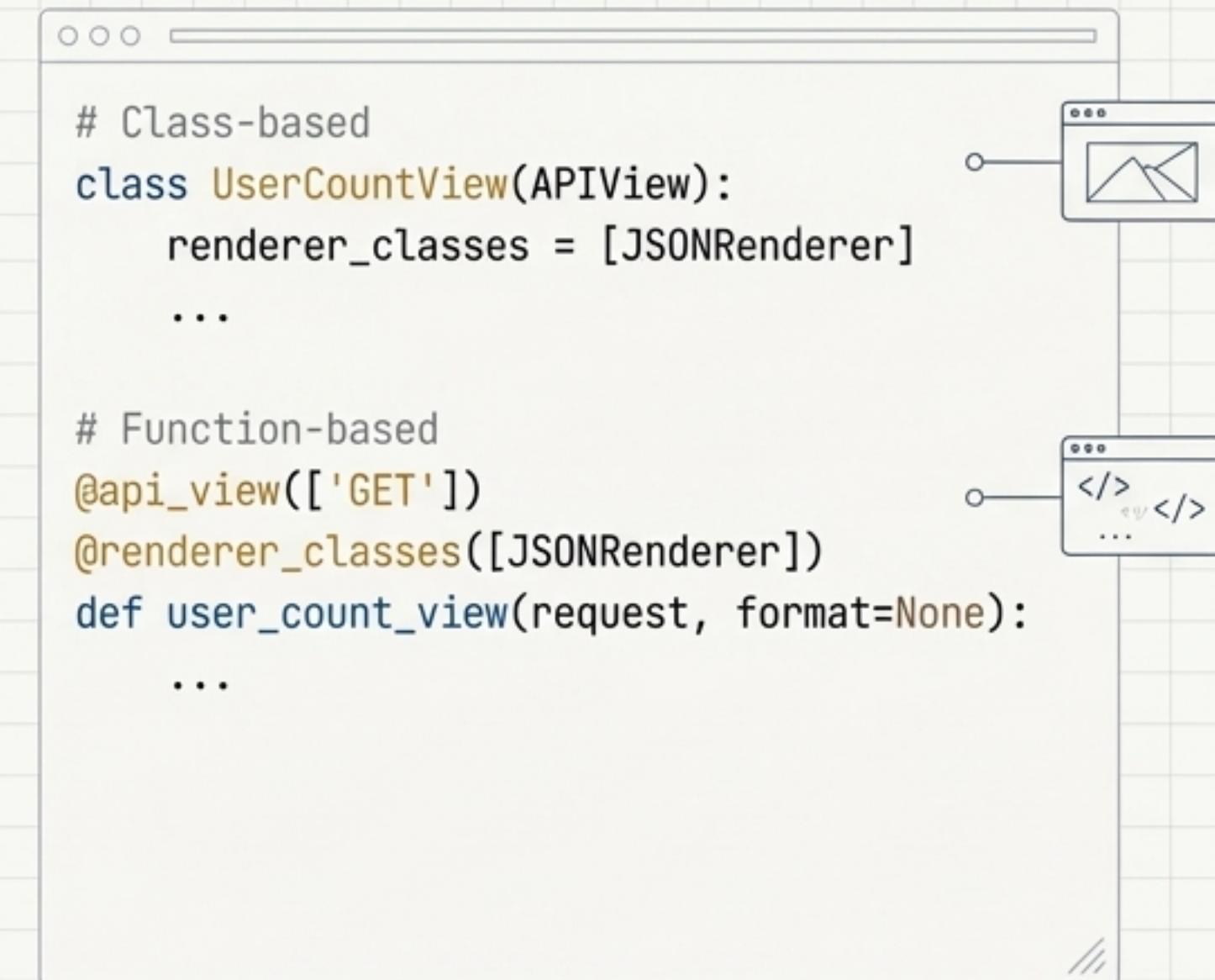
### نکته بسیار مهم: «ترتیب مهم است!»

اولین رندر در لیست به عنوان پیشفرض برای درخواست‌های مهم (مانند هدر (Accept: \*/\*\*/: Accept: \*/\*)) انتخاب می‌شود.



## پیکربندی برای هر View (کلاسی یا تابعی)

تنظیمات محلی بر تنظیمات سراسری اولویت دارند.



```
# Class-based  
class UserCountView(APIView):  
    renderer_classes = [JSONRenderer]  
    ...  
  
# Function-based  
@api_view(['GET'])  
@renderer_classes([JSONRenderer])  
def user_count_view(request, format=None):  
    ...
```

# `JSONRenderer` : جعبه ابزار داخلی (بخش ۱)

- هدف: رندر کردن داده به فرمت JSON با انکدینگ utf-8.
- مشخصات:
  - `media\_type`: application/json
  - `format`: 'json'
- تنظیمات مرتبط: اشاره به کلیدهای COMPACT\_JSON و UNICODE\_JSON در تنظیمات برای تغییر سبک انکدینگ انکدینگ پیشفرض.

ویژگی برجسته: خروجی تورفته (indented)

Compact (Default)

```
{"id":1,"username":"admin","email":"admin@example.com"}
```

Indented

```
{
  "id": 1,
  "username": "admin",
  "email": "admin@example.com"
}
```

Accept: application/json; indent=4

# جعبه ابزار داخلی (بخش ۲): رندرهای مبتنی بر HTML

## TemplateHTMLRenderer

- **هدف:** رندر کردن داده در قالب HTML با استفاده از تمپلیت‌های جنگو.
- **تفاوت کلیدی:** داده‌ها نیازی به سریالایز شدن ندارند و به عنوان Response به context ارسال می‌شوند. نیاز به تعیین template\_name دارد.

```
Response({'user': self.object},  
         template_name='user_detail.html')
```

نکته مهم: ترتیب اولویت برای پیدا کردن نام تمپلیت:

- (۱) آرگومان `template\_name` در Response
- (۲) اتربیوت template\_name در کلاس
- (۳) خروجی view.get\_template\_names()

## StaticHTMLRenderer

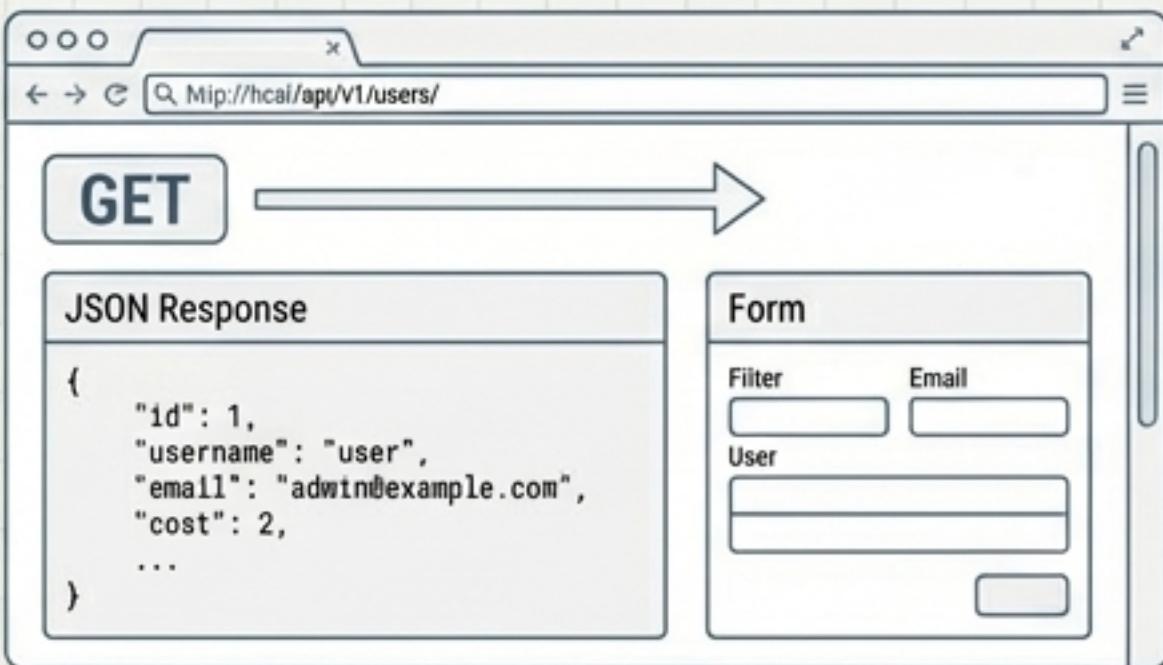
- **هدف:** بازگرداندن مستقیم یک رشته HTML از پیش رندر شده.
- **کاربرد:** برای صفحات HTML ساده و استاتیک که از طریق DRF سرو می‌شوند.

```
@api_view(['GET'])  
@renderer_classes([StaticHTMLRenderer])  
def simple_html_view(request):  
    data = '<html><body><h1>Hello, world</h1>  
          </body></html>'  
    return Response(data)
```

# جعبه ابزار داخلی (بخش ۳): رابطهای تعاملی

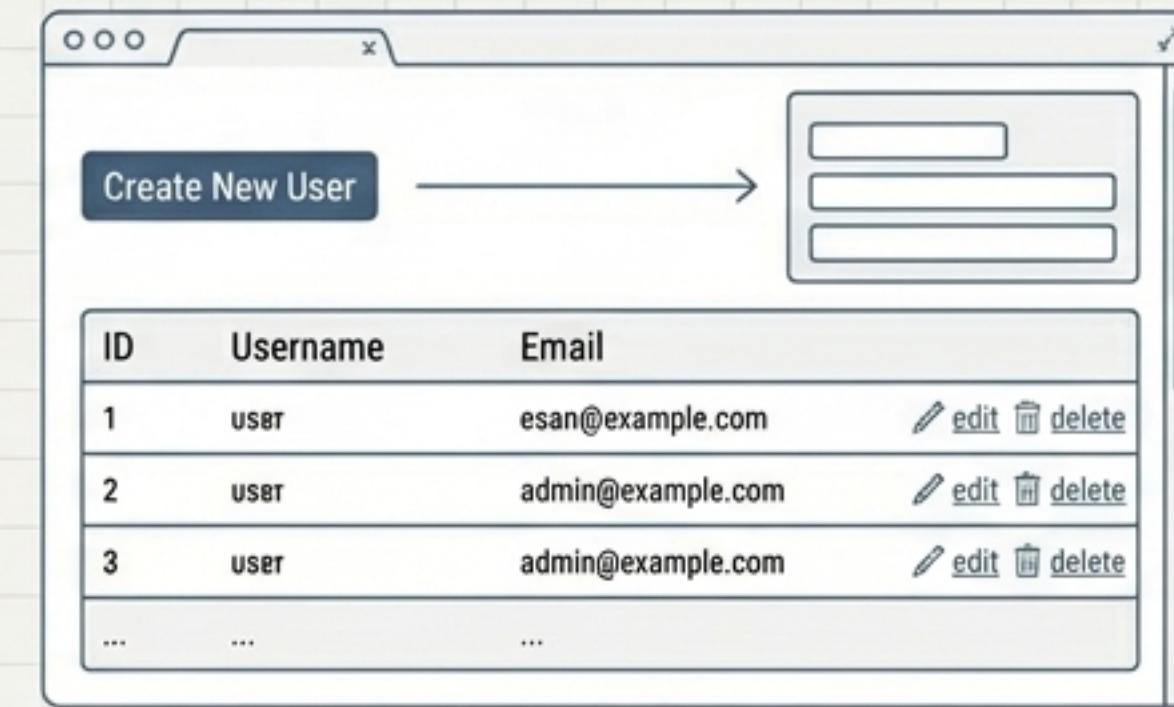
## ‘BrowsableAPIRenderer’

- هدف: ایجاد رابط کاربری HTML دوستانه و خود-مستندساز (self-documenting).
- نحوه کار: خروجی رندر بعدی با بالاترین اولویت را در یک صفحه HTML قرار می‌دهد.
- نکته پیشرفت: می‌توان با override کردن متدهای `get_default_renderer()` و `override()` رفتار آن را سفارشی کرد.

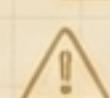


## ‘AdminRenderer’

- هدف: ارائه یک رابط کاربری شبیه به پنل ادمین جنگو برای عملیات CRUD.



نکات مهم: محدودیت در کار با سرالایزرهای تو در تو (nested) یا لیستی. برای ایجاد لینک به صفحات جزئیات، نیازمند فیلد `url` (یا فیلد تعریف شده در `URL\_FIELD\_NAME`) در داده‌ها است.



# جعبه ابزار داخلی (بخش ۴): رندرهای تخصصی و ابزاری

## `HTMLFormRenderer`

- هدف: رندر کردن یک سریالایزر به شکل یک فرم HTML (بدون تگ‌های `<form>` و دکمه `submit`).
- نحوه استفاده: به طور مستقیم استفاده نمی‌شود، بلکه از طریق تگ `{% render_form serializer %}` در تمپلیت‌ها به کار می‌رود.

```
{% load rest_framework %}  
<form action="/submit-report/" method="post">  
    {% csrf_token %}  
    {% render_form serializer %}  
    <input type="submit" value="Save" />  
</form>
```

## `MultiPartRenderer`

- هدف: برای ساخت درخواست‌های `.multipart/form-data`

## برای پاسخ (response) استفاده نمی‌شود

بلکه برای ایجاد درخواست‌های تستی با `test request factory` و `test client` به کار می‌رود.

# ساخت Renderer سفارشی: کنترل کامل در دستان شما

نقشه ساخت: از rest\_framework.renderers.BaseRenderer ارث بری کنید.

```
class YourRenderer(BaseRenderer):
    # جزء اصلی
    media_type = ...
    format = ...
```

def render(self, data, accepted\_media\_type, renderer\_context):  
 # ...  
 return bytestring

(`media\_type` : نوع MIME (مثلا: "text/plain").

`format` : پسوند URL (مثلا: "txt").

●: متدهای `render(...)` کار اصلی را انجام می‌دهد و

باید یک bytestring برگرداند.

## تشریح آرگومان‌های متدهای `render`

: داده‌های ارسال شده به (Response).

: نوع رسانه پذیرفته شده توسط کلاینت (می‌تواند شامل پارامتر باشد).

: یک دیکشنری شامل اطلاعات زمینه‌ای مانند view , request , response , args , kwargs .

# مثال عملی: `PlainTextRenderer`:

```
from django.utils.encoding import smart_str
from rest_framework import renderers

class PlainTextRenderer(renderers.BaseRenderer):
    media_type = 'text/plain'
    format = 'txt'
    charset = 'utf-8'

    def render(self, data, accepted_media_type=None,
              renderer_context=None):
        return smart_str(data, encoding=self.charset)
```

> GET /api/status.txt

HTTP/1.1 200 OK

Content-Type: text/plain; charset=utf-8

All systems operational.

# تنظیمات دقیق: `charset` و داده‌های باینری

## مجموعه کاراکتر (Character Set)

- پیش‌فرض `UTF-8` است.
- برای تغییر آن، اtribut `charset` را در کلاس رندر تنظیم کنید (مثال: `charset = 'iso-8859-1'`)

## داده‌های باینری (مانند تصاویر)

- برای جلوگیری از اضافه شدن `charset` به هدر `Content-Type` مقدار `charset = None` را تنظیم کنید.
- برای اینکه Browsable API سعی نکند محتوای باینری را به عنوان رشته نمایش دهد، `render\_style = 'binary'` را تنظیم کنید.

## مثال کامل: `JPEGRenderer`

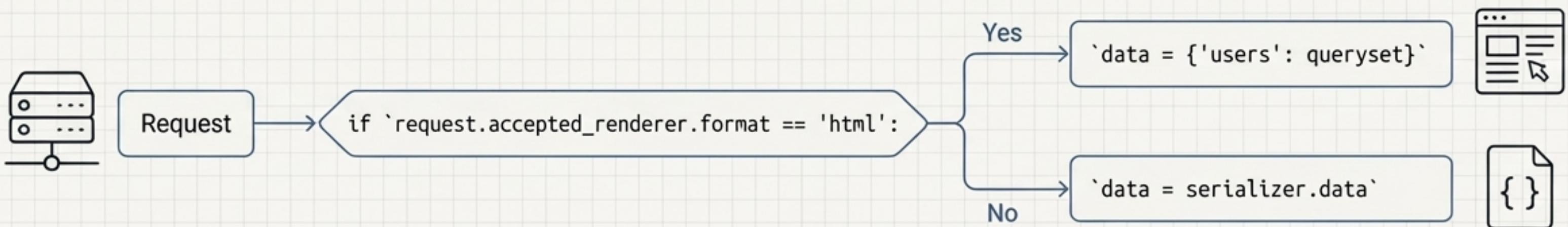
```
class JPEGRenderer(renderers.BaseRenderer):  
    media_type = 'image/jpeg'  
    format = 'jpg'  
    charset = None  
    render_style = 'binary'  
  
    def render(self, data, accepted_media_type=None,  
              renderer_context=None):  
        return data
```



# تکنیک‌های پیشرفت‌های اعطاف‌پذیری حد اکثری

## تغییر رفتار بر اساس Media Type

درون view، با دسترسی به `request.accepted\_renderer.format` ب `serializer.format` تغییر دهید.



## مشخص کردن ناقص (Underspecifying) Media Type

می‌توانید از `media\_type` عمومی مانند `'\*/\*'` یا حتی `image/\*` استفاده کنید تا به طیف وسیعی از انواع رسانه پاسخ دهید.

### نکته مهم:

در این حالت، باید هنگام بازگرداندن پاسخ، `content\_type` را به صراحت در آبجکت `Response` مشخص کنید (مثال: `(Response(data, content\_type='image/png'))`).

# تفکر معماری: طراحی Media Type و مدیریت خطا

## طراحی Media Type های سفارشی

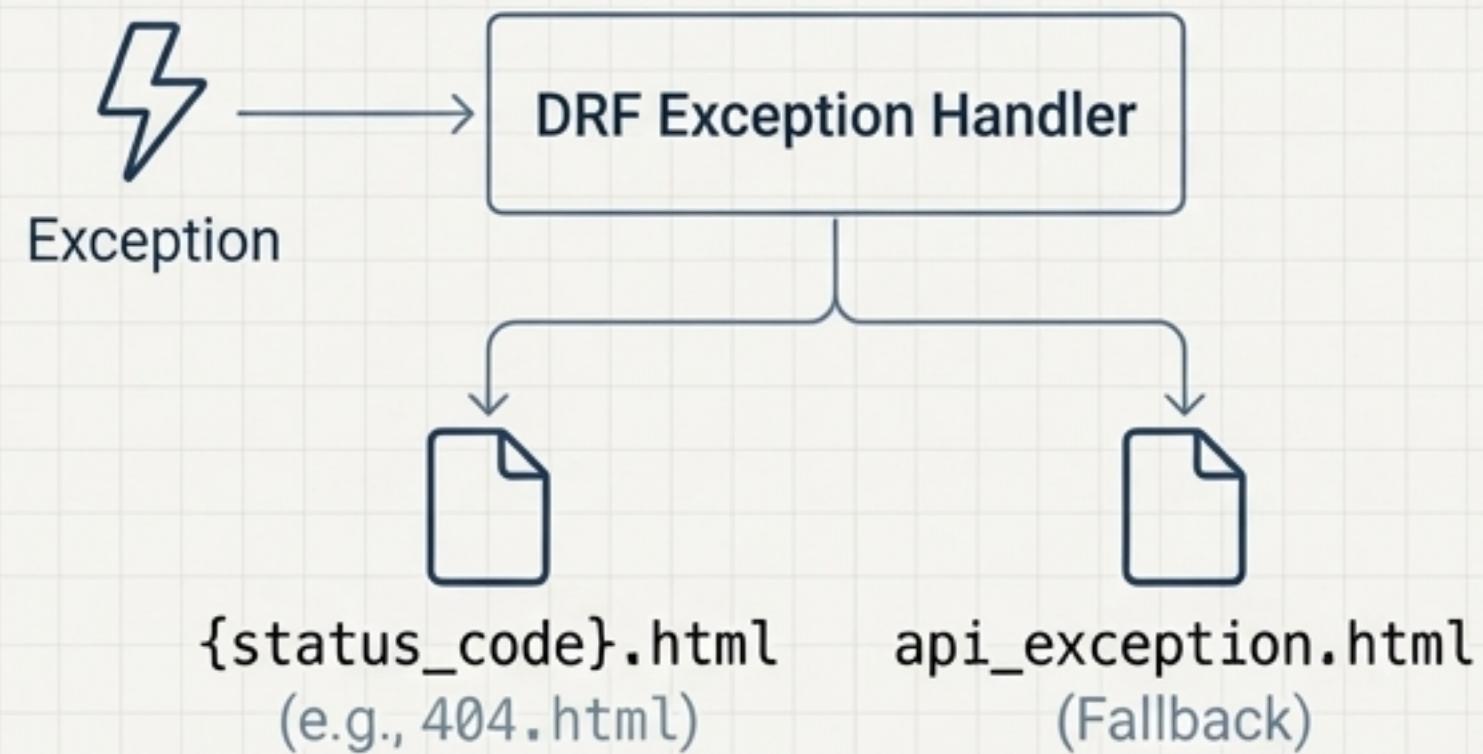
معرفی کوتاه مفهوم HATEOAS و انواع رسانه سفارشی (مثال: .(application/vnd.github+json))

“ A REST API should spend almost all of its descriptive effort in defining the media type(s) used for representing resources and driving application state... ”

Roy Fielding

## نماهای خطای HTML (HTML Error Views)

برای رندرهای HTML، در صورت بروز استثناء، DRF به ترتیب به دنبال تمپلیت‌های {status\_code}.html و سپس api\_exception.html می‌گردد.



اگر DEBUG=True باشد، صفحه خطای استاندارد جنگو نمایش داده می‌شود.

# اکوسیستم گسترده: پکیج‌های شخص ثالث (Third-Party)

فرمت‌های داده	صفحات گسترده و علم داده	اسناد
 YAML `djangorestframework-yaml`	 Microsoft Excel `drf-excel`	 LaTeX (PDF) `rest-framework-latex`
 XML `djangorestframework-xml`	 CSV `djangorestframework-csv`	
 JSONP `djangorestframework-jsonp` <small>(از CORS به عنوان جایگزین مدرن‌تر استفاده کنید)</small>	 Pandas `django-rest-pandas`	
 MessagePack `djangorestframework-msgpack`		
 UltraJSON `drf_ujson2`		
 CamelCase JSON `djangorestframework-camel-case`		

# جمع‌بندی و نکات کلیدی

۱. **کنترل کامل خروجی:** Rendererها مکانیسم DRF برای تبدیل داده برای هر نوع کلاینتی هستند. 
۲. **مذاکره، کلید تصمیم‌گیری:** هدر `Accept` و پسوندهای فرمت، خروجی نهایی را تعیین می‌کنند. 
۳. **جعبه ابزار غنی:** مجموعه‌ای جامع از رندرهای داخلی برای داده، HTML و API‌های تعاملی فراهم می‌کند. 
۴. **توسعه‌پذیری بی‌پایان:** اگر رندری وجود ندارد، می‌توانید با ساخت یک رندر سفارشی آن را ایجاد کنید. 
۵. **اکوسیستم پویا:** مجموعه وسیعی از پکیج‌های شخص ثالث، قابلیت‌های رندرینگ را فراتر می‌برند. 

سلط بر رندرها به معنای سلط بر صدای API شماست.