source: response.py

# Responses

> Unlike basic HttpResponse objects, TemplateResponse objects retain the details of the context that was provided by the view to compute the response. The final output of the response is not computed until it is needed, later in the response process.
>
> — Django documentation

REST framework supports HTTP content negotiation by providing a `Response` class which allows you to return content that can be rendered into multiple content types, depending on the client request.

The `Response` class subclasses Django's `SimpleTemplateResponse`. `Response` objects are initialised with data, which should consist of native Python primitives. REST framework then uses standard HTTP content negotiation to determine how it should render the final response content.

There's no requirement for you to use the `Response` class, you can also return regular `HttpResponse` or `StreamingHttpResponse` objects from your views if required. Using the `Response` class simply provides a nicer interface for returning content-negotiated Web API responses, that can be rendered to multiple formats.

Unless you want to heavily customize REST framework for some reason, you should always use an `APIView` class or `@api_view` function for views that return `Response` objects. Doing so ensures that the view can perform content negotiation and select the appropriate renderer for the response, before it is returned from the view.

---

# Creating responses

## Response()

**Signature:** `Response(data, status=None, template_name=None, headers=None, content_type=None)`

Unlike regular `HttpResponse` objects, you do not instantiate `Response` objects with rendered content. Instead you pass in unrendered data, which may consist of any Python primitives.

The renderers used by the `Response` class cannot natively handle complex datatypes such as Django model instances, so you need to serialize the data into primitive datatypes before creating the `Response` object.

You can use REST framework's `Serializer` classes to perform this data serialization, or use your own custom serialization.

Arguments:

- `data`: The serialized data for the response.
- `status`: A status code for the response. Defaults to 200. See also status codes.
- `template_name`: A template name to use if `HTMLRenderer` is selected.
- `headers`: A dictionary of HTTP headers to use in the response.
- `content_type`: The content type of the response. Typically, this will be set automatically by the renderer as determined by content negotiation, but there may be some cases where you need to specify the content type explicitly.

---

# Attributes

## .data

The unrendered content of a `Request` object.

## .status_code

The numeric status code of the HTTP response.

## .content

The rendered content of the response. The `.render()` method must have been called before `.content` can be accessed.

## .template_name

The `template_name`, if supplied. Only required if `HTMLRenderer` or some other custom template renderer is the accepted renderer for the response.

## .accepted_renderer

The renderer instance that will be used to render the response.

Set automatically by the `APIView` or `@api_view` immediately before the response is returned from the view.

## .accepted_media_type

The media type that was selected by the content negotiation stage.

Set automatically by the `APIView` or `@api_view` immediately before the response is returned from the view.

### .renderer_context

A dictionary of additional context information that will be passed to the renderer's `.render()` method.

Set automatically by the `APIView` or `@api_view` immediately before the response is returned from the view.

---

## Standard HttpResponse attributes

The `Response` class extends `SimpleTemplateResponse`, and all the usual attributes and methods are also available on the response. For example you can set headers on the response in the standard way:

```
response = Response()
response['Cache-Control'] = 'no-cache'
```

### .render()

**Signature:** `.render()`

As with any other `TemplateResponse`, this method is called to render the serialized data of the response into the final response content. When `.render()` is called, the response content will be set to the result of calling the `.render(data, accepted_media_type, renderer_context)` method on the `accepted_renderer` instance.

You won't typically need to call `.render()` yourself, as it's handled by Django's standard response cycle.