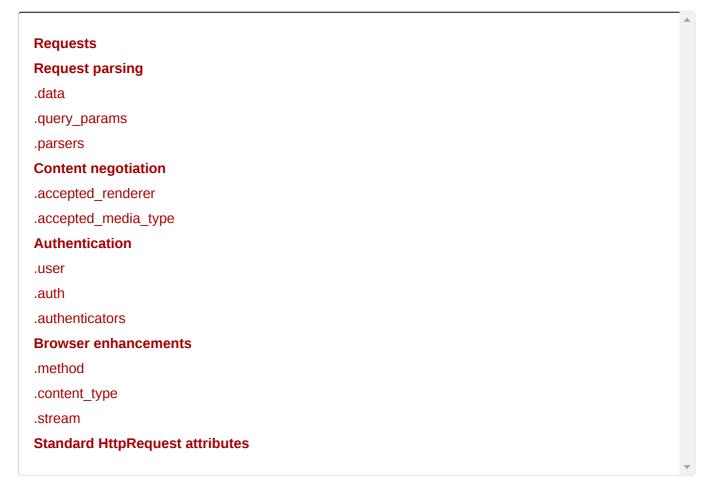
Django REST framework





request.py

Requests

— Malcom Tredinnick, Django developers group

REST framework's Request class extends the standard HttpRequest, adding support for REST framework's flexible request parsing and request authentication.

Request parsing

REST framework's Request objects provide flexible request parsing that allows you to treat requests with JSON data or other media types in the same way that you would normally deal with form data.

.data

request.data returns the parsed content of the request body. This is similar to the standard request.POST and request.FILES attributes except that:

- It includes all parsed content, including file and non-file inputs.
- It supports parsing the content of HTTP methods other than POST, meaning that you can access the content of PUT and PATCH requests.

 It supports REST framework's flexible request parsing, rather than just supporting form data. For example you can handle incoming JSON data in the same way that you handle incoming form data.

For more details see the parsers documentation.

.query_params

request.query_params is a more correctly named synonym for request.GET.

For clarity inside your code, we recommend using request.get instead of the Django's standard request.GET. Doing so will help keep your codebase more correct and obvious - any HTTP method type may include query parameters, not just GET requests.

.parsers

The APIView class or @api_view decorator will ensure that this property is automatically set to a list of Parser instances, based on the parser_classes set on the view or based on the DEFAULT_PARSER_CLASSES setting.

You won't typically need to access this property.

Note: If a client sends malformed content, then accessing request.data may raise a ParseError. By default REST framework's APIView class or @api_view decorator will catch the error and return a 400 Bad Request response.

If a client sends a request with a content-type that cannot be parsed then a UnsupportedMediaType exception will be raised, which by default will be caught and return a 415 Unsupported Media Type response.

Content negotiation

The request exposes some properties that allow you to determine the result of the content negotiation stage. This allows you to implement behaviour such as selecting a different serialisation schemes for different media types.

.accepted_renderer

The renderer instance what was selected by the content negotiation stage.

.accepted media type

A string representing the media type that was accepted by the content negotiation stage.

Authentication

REST framework provides flexible, per-request authentication, that gives you the ability to:

- · Use different authentication policies for different parts of your API.
- Support the use of multiple authentication policies.
- Provide both user and token information associated with the incoming request.

.user

request.user typically returns an instance of django.contrib.auth.models.User, although the behavior depends on the authentication policy being used.

If the request is unauthenticated the default value of request.user is an instance of django.contrib.auth.models.AnonymousUser.

For more details see the authentication documentation.

.auth

request.auth returns any additional authentication context. The exact behavior of request.auth depends on the authentication policy being used, but it may typically be an instance of the token that the request was authenticated against.

If the request is unauthenticated, or if no additional context is present, the default value of request.auth is None.

For more details see the authentication documentation.

.authenticators

The APIView class or @api_view decorator will ensure that this property is automatically set to a list of Authentication instances, based on the authentication_classes set on the view or based on the DEFAULT_AUTHENTICATORS setting.

You won't typically need to access this property.

Browser enhancements

REST framework supports a few browser enhancements such as browser-based PUT, PATCH and DELETE forms.

.method

request.method returns the uppercased string representation of the request's HTTP method.

Browser-based PUT, PATCH and DELETE forms are transparently supported.

For more information see the browser enhancements documentation.

.content_type

request.content_type, returns a string object representing the media type of the HTTP request's body, or an empty string if no media type was provided.

You won't typically need to directly access the request's content type, as you'll normally rely on REST framework's default request parsing behavior.

If you do need to access the content type of the request you should use the .content_type property in preference to using request.META.get('HTTP_CONTENT_TYPE'), as it provides transparent support for browser-based non-form content.

For more information see the browser enhancements documentation.

.stream

request.stream returns a stream representing the content of the request body.

You won't typically need to directly access the request's content, as you'll normally rely on REST framework's default request parsing behavior.

Standard HttpRequest attributes

As REST framework's Request extends Django's HttpRequest, all the other standard attributes and methods are also available. For example the request.META and request.session dictionaries are available as normal.

Note that due to implementation reasons the Request class does not inherit from HttpRequest class, but instead extends the class using composition.

Documentation built with MkDocs.