

Metadata

Setting the metadata scheme

Creating schema endpoints

Custom metadata classes

Example

Third party packages

DRF-schema-adapter

metadata.py

Metadata

“ [The `OPTIONS`] method allows a client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval.

— RFC7231, Section 4.3.7.

REST framework includes a configurable mechanism for determining how your API should respond to `OPTIONS` requests. This allows you to return API schema or other resource information.

There are not currently any widely adopted conventions for exactly what style of response should be returned for HTTP `OPTIONS` requests, so we provide an ad-hoc style that returns some useful information.

Here's an example response that demonstrates the information that is returned by default.

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json

{
    "name": "To Do List",
    "description": "List existing 'To Do' items, or create a new item.",
    "renders": [
        "application/json",
        "text/html"
    ],
    "parses": [
        "application/json",
        "application/x-www-form-urlencoded",
        "multipart/form-data"
    ],
    "actions": {
        "POST": {
            "note": {
                "type": "string",
                ...
            }
        }
    }
}
```

```
        "required": false,
        "read_only": false,
        "label": "title",
        "max_length": 100
    }
}
}
```

Setting the metadata scheme

You can set the metadata class globally using the `'DEFAULT_METADATA_CLASS'` settings key:

```
REST_FRAMEWORK = {
    'DEFAULT_METADATA_CLASS': 'rest_framework.metadata.SimpleMetadata'
}
```

Or you can set the metadata class individually for a view:

```
class APIRoot(APIView):
    metadata_class = APIRootMetadata

    def get(self, request, format=None):
        return Response({
            ...
        })
```

The REST framework package only includes a single metadata class implementation, named `SimpleMetadata`. If you want to use an alternative style you'll need to implement a custom metadata class.

Creating schema endpoints

If you have specific requirements for creating schema endpoints that are accessed with regular `GET` requests, you might consider re-using the metadata API for doing so.

For example, the following additional route could be used on a viewset to provide a linkable schema endpoint.

```
@list_route(methods=['GET'])
def schema(self, request):
    meta = self.metadata_class()
    data = meta.determine_metadata(request, self)
    return Response(data)
```

There are a couple of reasons that you might choose to take this approach, including that `OPTIONS` responses **are not cacheable**.

Custom metadata classes

If you want to provide a custom metadata class you should override `BaseMetadata` and implement the `determine_metadata(self, request, view)` method.

Useful things that you might want to do could include returning schema information, using a format such as `JSON schema`, or returning debug information to admin users.

Example

The following class could be used to limit the information that is returned to `OPTIONS` requests.

```
class MinimalMetadata(BaseMetadata):
    """
    Don't include field and other information for `OPTIONS` requests.
    Just return the name and description.
    """
    def determine_metadata(self, request, view):
        return {
            'name': view.get_view_name(),
            'description': view.get_view_description()
        }
```

Then configure your settings to use this custom class:

```
REST_FRAMEWORK = {
    'DEFAULT_METADATA_CLASS': 'myproject.apps.core.MinimalMetadata'
}
```

Third party packages

The following third party packages provide additional metadata implementations.

DRF-schema-adapter

`drf-schema-adapter` is a set of tools that makes it easier to provide schema information to frontend frameworks and libraries. It provides a metadata mixin as well as 2 metadata classes and several adapters suitable to generate `json-schema` as well as schema information readable by various libraries.

You can also write your own adapter to work with your specific frontend. If you wish to do so, it also provides an exporter that can export those schema information to json files.

