pagination.py

Pagination

> *Django provides a few classes that help you manage paginated data – that is, data that's split across several pages, with "Previous/Next" links.*
>
> — *Django documentation*

REST framework includes support for customizable pagination styles. This allows you to modify how large result sets are split into individual pages of data.

The pagination API can support either:

- Pagination links that are provided as part of the content of the response.
- Pagination links that are included in response headers, such as `Content-Range` or `Link`.

The built-in styles currently all use links included as part of the content of the response. This style is more accessible when using the browsable API.

Pagination is only performed automatically if you're using the generic views or viewsets. If you're using a regular `APIView`, you'll need to call into the pagination API yourself to ensure you return a paginated response. See the source code for the `mixins.ListModelMixin` and `generics.GenericAPIView` classes for an example.

Pagination can be turned off by setting the pagination class to `None`.

Setting the pagination style

The default pagination style may be set globally, using the `DEFAULT_PAGINATION_CLASS` and `PAGE_SIZE` setting keys. For example, to use the built-in limit/offset pagination, you would do something like this:

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.LimitOffsetPagination',
    'PAGE_SIZE': 100
}
```

Note that you need to set both the pagination class, and the page size that should be used.

You can also set the pagination class on an individual view by using the `pagination_class` attribute. Typically you'll want to use the same pagination style throughout your API, although you might want to vary individual aspects of the pagination, such as default or maximum page size, on a per-view basis.

Modifying the pagination style

If you want to modify particular aspects of the pagination style, you'll want to override one of the pagination classes, and set the attributes that you want to change.

```
class LargeResultsSetPagination(PageNumberPagination):
    page_size = 1000
    page_size_query_param = 'page_size'
    max_page_size = 10000

class StandardResultsSetPagination(PageNumberPagination):
    page_size = 100
    page_size_query_param = 'page_size'
    max_page_size = 1000
```

You can then apply your new style to a view using the `.pagination_class` attribute:

```
class BillingRecordsView(generics.ListAPIView):
    queryset = Billing.objects.all()
    serializer_class = BillingRecordsSerializer
    pagination_class = LargeResultsSetPagination
```

Or apply the style globally, using the `DEFAULT_PAGINATION_CLASS` settings key. For example:

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'apps.core.pagination.StandardResultsSetPagination'
}
```

API Reference

PageNumberPagination

This pagination style accepts a single number page number in the request query parameters.

Request:

```
GET https://api.example.org/accounts/?page=4
```

Response:

```
HTTP 200 OK
{
    "count": 1023
    "next": "https://api.example.org/accounts/?page=5",
    "previous": "https://api.example.org/accounts/?page=3",
    "results": [
       …
    ]
}
```

Setup

To enable the `PageNumberPagination` style globally, use the following configuration, modifying the `PAGE_SIZE` as desired:

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 100
}
```

On `GenericAPIView` subclasses you may also set the `pagination_class` attribute to select `PageNumberPagination` on a per-view basis.

Configuration

The `PageNumberPagination` class includes a number of attributes that may be overridden to modify the pagination style.

To set these attributes you should override the `PageNumberPagination` class, and then enable your custom pagination class as above.

- `django_paginator_class` - The Django Paginator class to use. Default is `django.core.paginator.Paginator`, which should be fine for most use cases.
- `page_size` - A numeric value indicating the page size. If set, this overrides the `PAGE_SIZE` setting. Defaults to the same value as the `PAGE_SIZE` settings key.

- `page_query_param` - A string value indicating the name of the query parameter to use for the pagination control.
- `page_size_query_param` - If set, this is a string value indicating the name of a query parameter that allows the client to set the page size on a per-request basis. Defaults to `None`, indicating that the client may not control the requested page size.
- `max_page_size` - If set, this is a numeric value indicating the maximum allowable requested page size. This attribute is only valid if `page_size_query_param` is also set.
- `last_page_strings` - A list or tuple of string values indicating values that may be used with the `page_query_param` to request the final page in the set. Defaults to `('last',)`
- `template` - The name of a template to use when rendering pagination controls in the browsable API. May be overridden to modify the rendering style, or set to `None` to disable HTML pagination controls completely. Defaults to `"rest_framework/pagination/numbers.html"`.

LimitOffsetPagination

This pagination style mirrors the syntax used when looking up multiple database records. The client includes both a "limit" and an "offset" query parameter. The limit indicates the maximum number of items to return, and is equivalent to the `page_size` in other styles. The offset indicates the starting position of the query in relation to the complete set of unpaginated items.

Request:

```
GET https://api.example.org/accounts/?limit=100&offset=400
```

Response:

```
HTTP 200 OK
{
    "count": 1023
    "next": "https://api.example.org/accounts/?limit=100&offset=500",
    "previous": "https://api.example.org/accounts/?limit=100&offset=300",
    "results": [
       …
    ]
}
```

Setup

To enable the `LimitOffsetPagination` style globally, use the following configuration:

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.LimitOffsetPagination'
}
```

Optionally, you may also set a `PAGE_SIZE` key. If the `PAGE_SIZE` parameter is also used then the `limit` query parameter will be optional, and may be omitted by the client.

On `GenericAPIView` subclasses you may also set the `pagination_class` attribute to select `LimitOffsetPagination` on a per-view basis.

Configuration

The `LimitOffsetPagination` class includes a number of attributes that may be overridden to modify the pagination style.

To set these attributes you should override the `LimitOffsetPagination` class, and then enable your custom pagination class as above.

- `default_limit` - A numeric value indicating the limit to use if one is not provided by the client in a query parameter. Defaults to the same value as the `PAGE_SIZE` settings key.
- `limit_query_param` - A string value indicating the name of the "limit" query parameter. Defaults to `'limit'`.
- `offset_query_param` - A string value indicating the name of the "offset" query parameter. Defaults to `'offset'`.
- `max_limit` - If set this is a numeric value indicating the maximum allowable limit that may be requested by the client. Defaults to `None`.
- `template` - The name of a template to use when rendering pagination controls in the browsable API. May be overridden to modify the rendering style, or set to `None` to disable HTML pagination controls completely. Defaults to `"rest_framework/pagination/numbers.html"`.

CursorPagination

The cursor-based pagination presents an opaque "cursor" indicator that the client may use to page through the result set. This pagination style only presents forward and reverse controls, and does not allow the client to navigate to arbitrary positions.

Cursor based pagination requires that there is a unique, unchanging ordering of items in the result set. This ordering might typically be a creation timestamp on the records, as this presents a consistent ordering to paginate against.

Cursor based pagination is more complex than other schemes. It also requires that the result set presents a fixed ordering, and does not allow the client to arbitrarily index into the result set. However it does provide the following benefits:

- Provides a consistent pagination view. When used properly `CursorPagination` ensures that the client will never see the same item twice when paging through records, even when new items are being inserted by other clients during the pagination process.
- Supports usage with very large datasets. With extremely large datasets pagination using offset-based pagination styles may become inefficient or unusable. Cursor based pagination schemes instead have fixed-time properties, and do not slow down as the dataset size increases.

Details and limitations

Proper use of cursor based pagination requires a little attention to detail. You'll need to think about what ordering you want the scheme to be applied against. The default is to order by `"-created"`. This assumes that **there must be a 'created' timestamp field** on the model instances, and will present a "timeline" style paginated view, with the most recently added items first.

You can modify the ordering by overriding the `'ordering'` attribute on the pagination class, or by using the `OrderingFilter` filter class together with `CursorPagination`. When used with `OrderingFilter` you should strongly consider restricting the fields that the user may order by.

Proper usage of cursor pagination should have an ordering field that satisfies the following:

- Should be an unchanging value, such as a timestamp, slug, or other field that is only set once, on creation.
- Should be unique, or nearly unique. Millisecond precision timestamps are a good example. This implementation of cursor pagination uses a smart "position plus offset" style that allows it to properly support not-strictly-unique values as the ordering.
- Should be a non-nullable value that can be coerced to a string.
- The field should have a database index.

Using an ordering field that does not satisfy these constraints will generally still work, but you'll be losing some of the benefits of cursor pagination.

For more technical details on the implementation we use for cursor pagination, the "Building cursors for the Disqus API" blog post gives a good overview of the basic approach.

Setup

To enable the `CursorPagination` style globally, use the following configuration, modifying the `PAGE_SIZE` as desired:

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.CursorPagination',
    'PAGE_SIZE': 100
}
```

On `GenericAPIView` subclasses you may also set the `pagination_class` attribute to select `CursorPagination` on a per-view basis.

Configuration

The `CursorPagination` class includes a number of attributes that may be overridden to modify the pagination style.

To set these attributes you should override the `CursorPagination` class, and then enable your custom pagination class as above.

- `page_size` = A numeric value indicating the page size. If set, this overrides the `PAGE_SIZE` setting. Defaults to the same value as the `PAGE_SIZE` settings key.
- `cursor_query_param` = A string value indicating the name of the "cursor" query parameter. Defaults to `'cursor'`.
- `ordering` = This should be a string, or list of strings, indicating the field against which the cursor based pagination will be applied. For example: `ordering = 'slug'`. Defaults to `-created`. This value may also be overridden by using `OrderingFilter` on the view.
- `template` = The name of a template to use when rendering pagination controls in the browsable API. May be overridden to modify the rendering style, or set to `None` to disable HTML pagination controls completely. Defaults to `"rest_framework/pagination/previous_and_next.html"`.

Custom pagination styles

To create a custom pagination serializer class you should subclass `pagination.BasePagination` and override the `paginate_queryset(self, queryset, request, view=None)` and `get_paginated_response(self, data)` methods:

- The `paginate_queryset` method is passed the initial queryset and should return an iterable object that contains only the data in the requested page.
- The `get_paginated_response` method is passed the serialized page data and should return a `Response` instance.

Note that the `paginate_queryset` method may set state on the pagination instance, that may later be used by the `get_paginated_response` method.

Example

Suppose we want to replace the default pagination output style with a modified format that includes the next and previous links under in a nested 'links' key. We could specify a custom pagination class like so:

```
class CustomPagination(pagination.PageNumberPagination):
    def get_paginated_response(self, data):
        return Response({
            'links': {
               'next': self.get_next_link(),
               'previous': self.get_previous_link()
            },
            'count': self.page.paginator.count,
            'results': data
        })
```

We'd then need to setup the custom class in our configuration:

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'my_project.apps.core.pagination.CustomPagination',
    'PAGE_SIZE': 100
}
```

Note that if you care about how the ordering of keys is displayed in responses in the browsable API you might choose to use an `OrderedDict` when constructing the body of paginated responses, but this is optional.

Header based pagination

Let's modify the built-in `PageNumberPagination` style, so that instead of include the pagination links in the body of the response, we'll instead include a `Link` header, in a similar style to the GitHub API.

```
class LinkHeaderPagination(pagination.PageNumberPagination):
```

```
    def get_paginated_response(self, data):
        next_url = self.get_next_link()
        previous_url = self.get_previous_link()

        if next_url is not None and previous_url is not None:
            link = '<{next_url}>; rel="next", <{previous_url}>; rel="prev"'
        elif next_url is not None:
            link = '<{next_url}>; rel="next"'
        elif previous_url is not None:
            link = '<{previous_url}>; rel="prev"'
        else:
            link = ''

        link = link.format(next_url=next_url, previous_url=previous_url)
        headers = {'Link': link} if link else {}

        return Response(data, headers=headers)
```

Using your custom pagination class

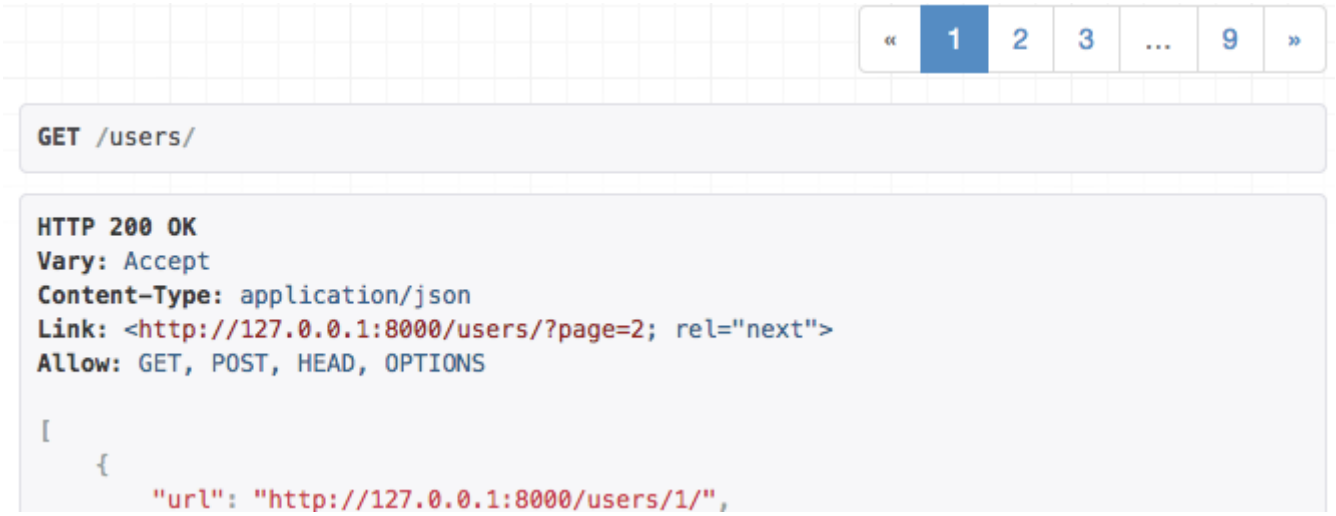
To have your custom pagination class be used by default, use the `DEFAULT_PAGINATION_CLASS` setting:

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'my_project.apps.core.pagination.LinkHeaderPagination',
    'PAGE_SIZE': 100
}
```

API responses for list endpoints will now include a `Link` header, instead of including the pagination links as part of the body of the response, for example:

Pagination & schemas

You can also make the pagination controls available to the schema autogeneration that REST framework provides, by implementing a `get_schema_fields()` method, which should return a list of `coreapi.Field` instances.



A custom pagination style, using the 'Link' header'

HTML pagination controls

By default using the pagination classes will cause HTML pagination controls to be displayed in the browsable API. There are two built-in display styles. The `PageNumberPagination` and `LimitOffsetPagination` classes display a list of page numbers with previous and next controls. The `CursorPagination` class displays a simpler style that only displays a previous and next control.

Customizing the controls

You can override the templates that render the HTML pagination controls. The two built-in styles are:

- `rest_framework/pagination/numbers.html`
- `rest_framework/pagination/previous_and_next.html`

Providing a template with either of these paths in a global template directory will override the default rendering for the relevant pagination classes.

Alternatively you can disable HTML pagination controls completely by subclassing on of the existing classes, setting `template = None` as an attribute on the class. You'll then need to configure your `DEFAULT_PAGINATION_CLASS` settings key to use your custom class as the default pagination style.

Low-level API

The low-level API for determining if a pagination class should display the controls or not is exposed as a `display_page_controls` attribute on the pagination instance. Custom pagination classes should be set to `True` in the `paginate_queryset` method if they require the HTML pagination controls to be displayed.

The `.to_html()` and `.get_html_context()` methods may also be overridden in a custom pagination class in order to further customize how the controls are rendered.

Third party packages

The following third party packages are also available.

DRF-extensions

The `DRF-extensions` package includes a `PaginateByMaxMixin` mixin class that allows your API clients to specify `?page_size=max` to obtain the maximum allowed page size.

drf-proxy-pagination

The `drf-proxy-pagination` package includes a `ProxyPagination` class which allows to choose pagination class with a query parameter.
