Django ezTables Documentation

Release 0.2.1

Axel Haustant

February 08, 2013

CONTENTS

Django ezTables provides easy integration between jQuery DataTables and Django.

CHAPTER

ONE

COMPATIBILITY

Django ezTables requires Python 2.7, Django 1.4 and Django.js 0.5.

CHAPTER

TWO

INSTALLATION

You can install Django ezTables with pip:

```
$ pip install django-eztables
```

or with easy_install:

```
$ easy_install django-eztables
```

Add `djangojs` and `django-eztables` to your `settings.INSTALLED_APPS`.

CHAPTER

THREE

FEATURES

- Datatables.net, plugins and localization integration with Django.
- **Server-side processing with a simple view supporting:**
  - sorting (single and multi columns)
  - filtering with regex support (global and by column)
  - formatting using format pattern
- Deferred loading support.
- Twitter Bootstrap integration.

CHAPTER

FOUR

DEMO

You can try the demo by cloning this repository and running the test server with provided data:

```
$ python manage.py syncdb
$ python manage.py loaddata eztables/demo/fixtures/browsers.json
$ pyhton manage.py runserver
```

Then open your browser to http://localhost:8000

CHAPTER

FIVE

DOCUMENTATION

5.1 Template tags

5.1.1 Initialization

You can either:

- load the template tag lib into each template manually:

```
{% load eztables %}
```

- load the template tag lib by adding to your `views.py`:

```
from django.template import add_to_builtins

add_to_builtins('eztables.templatetags.eztables')
```

5.1.2 Usage

datatables_js

A `{% datatables_js %}` tag is available to include the datatables javascript library. After loading, you can use the Datatables library as usual:

```
<table id="my-table">
</table>

{% datatables_js %}
<script>
    $('#my-table').dataTable();
</script>
```

bootstrap support

If you want to use the twitter bootstrap style (based on this blog post), 2 template tags are provided for:

- `{% datatables_bootstrap_js %}` for the javascript part.
- `{% datatables_bootstrap_css %}` for the css part.

```
<head>
    {% datatables_bootstrap_css %}

    {% datatables_js %}
    {% datatables_bootstrap_js %}
</head>

<table id="my-table">
</table>

<script>
    $('#my-table').dataTable({
        "bPaginate": true,
        "sPaginationType": "bootstrap",
        "bScrollCollapse": true
    });
</script>
```

5.2 Server-side processing

Django ezTable provide a single view to implement server-side pagination: `eztables.views.DatatablesView`.

It follows the Django Class-based Views pattern and can render Array-based or Object-based JSON.

As it extends `django.views.generic.list.MultipleObjectMixin` it expects the `model` attribute to be set in both case.

Both modes expect a `fields` attribute that can optionnaly contains format patterns.

The exemple will use the same models as the demo:

```
from django.db import models


class Engine(models.Model):
    name = models.CharField(max_length=128)
    version = models.CharField(max_length=8, blank=True)
    css_grade = models.CharField(max_length=3)

    def __unicode__(self):
        return '%s %s (%s)' % (self.name, self.version or '-', self.css_grade)


class Browser(models.Model):
    name = models.CharField(max_length=128)
    platform = models.CharField(max_length=128)
    version = models.CharField(max_length=8, blank=True)
    engine = models.ForeignKey(Engine)

    def __unicode__(self):
        return '%s %s' % (self.name, self.version or '-')
```

5.2.1 Array-based JSON

To render an array-based JSON, you must provide `fields` as a `list` or a `tuple` containing the field names.

```
from eztables.views import DatatablesView
from myapp.models import Browser

class BrowserDatatablesView(DatatablesView):
    model = Browser
    fields = (
        'engine__name',
        'name',
        'platform',
        'engine__version',
        'engine__css_grade',
    )
```

You can simply instanciate your datatable with:

```
$(function(){
    $('#browser-table').dataTable({
        "bPaginate": true,
        "sPaginationType": "bootstrap",
        "bProcessing": true,
        "bServerSide": true,
        "sAjaxSource": Django.url('dt-browsers-default')
    });
});
```

5.2.2 Object-based JSON

To render an array-based JSON, you must provide `fields` as a `dict` containing the mapping between the JSON fields names and the model fields.

```
from eztables.views import DatatablesView
from myapp.models import Browser

class ObjectBrowserDatatablesView(DatatablesView):
    model = Browser
    fields = {
        'name': 'name',
        'engine': 'engine__name',
        'platform': 'platform',
        'engine_version': 'engine__version',
        'css_grade': 'engine__css_grade',
    }
```

You need to use the `aoColumns` properties in the DataTables initialization:

```
$(function(){
    $('#browser-table').dataTable({
        "bPaginate": true,
        "sPaginationType": "bootstrap",
        "bProcessing": true,
        "bServerSide": true,
        "sAjaxSource": Django.url('dt-browsers-objects'),
        "aoColumns": [
            { "mData": "engine" },
            { "mData": "name" },
            { "mData": "platform" },
            { "mData": "engine_version" },
```

```
            { "mData": "css_grade" }
        ]
    });
});
```

5.2.3 Format patterns

You can optionnaly provide some format patterns in the field definition:

```
from eztables.views import DatatablesView
from myapp.models import Browser

class FormattedBrowserDatatablesView(DatatablesView):
    model = Browser
    fields = (
        'engine__name',
        '{name} {version}',
        'platform',
        'engine__version',
        'engine__css_grade',
    )

class FormattedObjectBrowserDatatablesView(DatatablesView):
    model = Browser
    fields = {
        'name': '{name} {version}',
        'engine': 'engine__name',
        'platform': 'platform',
        'engine_version': 'engine__version',
        'css_grade': 'engine__css_grade',
    }
```

5.2.4 Custom sort

You can implement a custom sort method. It have to be named `sort_col_X` where `X` should be the index given by the datatables request (correspond to the filtered column).

It take the requested direction (`''` or `'-'`) as parameter and should return one or more Django order statement.

```
class CustomSortBrowserDatatablesView(BrowserDatatablesView):

    def sort_col_1(self, direction):
        '''Sort on version instead of name'''
        return '%sversion' % direction

    def sort_col_2(self, direction):
        '''Sort on name and platform instead of platform'''
        return ('%sname' % direction, '%splatform' % direction)
```

5.2.5 Custom search

You can implement a custom search method. It have to be named `search_col_X` where `X` should be the index given by the datatables request (correspond to the filtered column).

It takes the search term and the queryset to filter as parameter and should return the filtered queryset.

```
class CustomSearchBrowserDatatablesView(BrowserDatatablesView):

    def search_col_1(self, search, queryset):
        '''Search on version instead of name'''
        return queryset.filter(version__icontains=search)
```

5.3 Localization

Django ezTable embed DataTables localizations.

Thay have the following naming convention:

```
{{STATIC_URL}}/js/libs/datatables/language.{{LANG}}.json
```

You can simply retrieve them with django.js:

```
$('#my-table').dataTable({
    ...
    "oLanguage": {
        "sUrl": Django.file("js/libs/datatables/language.fr.json")
    }
    ...
});
```

You can obtain the current language code with django.js too:

```
var code = Django.context.LANGUAGE_CODE;
```

Be carefull, no localization is provided for English language.

5.4 API

5.4.1 `eztables` – Main package

5.4.2 `eztables.forms` – DataTables pagination form processing

class `eztables.forms.`**`DatatablesForm`**(**args*, ***kwargs*)

Bases: `django.forms.forms.Form`

Datatables server side processing Form

See: http://www.datatables.net/usage/server-side

`bRegex` = None
: True if the global filter should be treated as a regular expression for advanced filtering, false if not.

`iColumns` = None
: Number of columns being displayed (useful for getting individual column search info)

`iDisplayLength` = None
: Number of records that the table can display in the current draw. It is expected that the number of records returned will be equal to this number, unless the server has fewer records to return.

`iDisplayStart` = None
: Display start point in the current data set.

iSortingCols = None
Number of columns to sort on

sEcho = None
Information for DataTables to use for rendering.

sSearch = None
Global search field

5.4.3 eztables.views – DataTables server-side processing view

class eztables.views.**DatatablesView**(***kwargs*)
Bases: django.views.generic.list.MultipleObjectMixin, django.views.generic.base.View

Render a paginated server-side Datatables JSON view.

See: http://www.datatables.net/usage/server-side

column_search(*queryset*)
Filter a queryset with column search

get_orders()
Get ordering fields for QuerySet.order_by

get_page(*form*)
Get the requested page

get_queryset()
Apply Datatables sort and search criterion to QuerySet

get_row(*row*)
Format a single row (if necessary)

get_rows(*rows*)
Format all rows

global_search(*queryset*)
Filter a queryset with global search

render_to_response(*form*, ***kwargs*)
Render Datatables expected JSON format

5.4.4 eztables.templatetags.eztables – Template tags

5.5 Changelog

5.5.1 0.2.1 (2013-02-08)

- Fix formatting with unicode

5.5.2 0.2 (2013-02-07)

- Handle custom server-side search implementation
- Handle custom server-side sort implementation

5.5.3 0.1.2 (2013-02-07)

- Fix static files packaging

5.5.4 0.1.1 (2013-02-07)

- Fix requirements packaging

5.5.5 0.1 (2013-02-07)

- Initial implementation

CHAPTER

SIX

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

e

eztables, ??
eztables.forms, ??
eztables.views, ??