django-listable Documentation

Release 0.4.3

Randle Taylor

Sep 25, 2017

Contents

Contents:

CHAPTER 1

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/randlet/django-listable/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

Write Documentation

django-listable could always use more documentation, whether as part of the official django-listable docs, in doc-strings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/randlet/django-listable/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *django-listable* for local development.

1. Fork the *django-listable* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:randlet/django-listable.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv django-listable
$ cd django-listable/
$ pip install -r requirements/test.txt
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 django-listable tests
    $ python setup.py test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/randlet/django-listable/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ python -m unittest tests.test_django_listable easytables
```

CHAPTER 2

Credits

Development Lead

- Randle Taylor <randle.taylor@gmail.com>

Contributors

None yet. Why not be the first?

CHAPTER 3

History

Fix values_to_dt to allow unicode

Add fix for when using FORCE_SCRIPT_NAME setting

Update to support Django 1.8-1.10 and Python 2.7-3.5

Fix formatting bug introduced by 0.3.8

0.3.8 (2016-09-27)

Fix unicode encoding error

0.3.7 (2016-08-25)

Add date range picker

0.3.6 (2016-06-29)

Add multi select and date select widgets (thanks to @ryanbottema)

0.3.5 (2016-06-22)

Fix filtering and count queries for django-mssql

0.3.3 (2015-04-12)

- Fix filtering of None values for SELECT fields

0.3.1 (2015-02-25)

- Fix issue with boolean field filtering

0.2.10 (2014-12-16)

- Fix issue with pagination type

0.2.9 (2014-12-15)

- Fix issue with namespaced urls

0.2.6 (2014-10-30)

- add view args & kwargs to context to allow full reverse

0.2.5 (2014-10-30)

- fix order_by

0.2.0 (2014-10-29)

- Complete overhaul of api

0.1.2 (2014-07-09)

- Fix saveState bug

0.1.0 (2013-08-15)

- First release on PyPI.

CHAPTER 4

TODO

- Implement defered loading of first page of results by rendering the table on the intitial page load (set iDeferLoading in datatables option).

CHAPTER 5

About

Listable is a Django package to make the integration of your Django models with Datatables.js easy.

Django-listable was motivated by my repeated need to generate sortable and filterable tables from my Django models for CRUD apps.

The idea is that you should easily be able to go from a model like this:

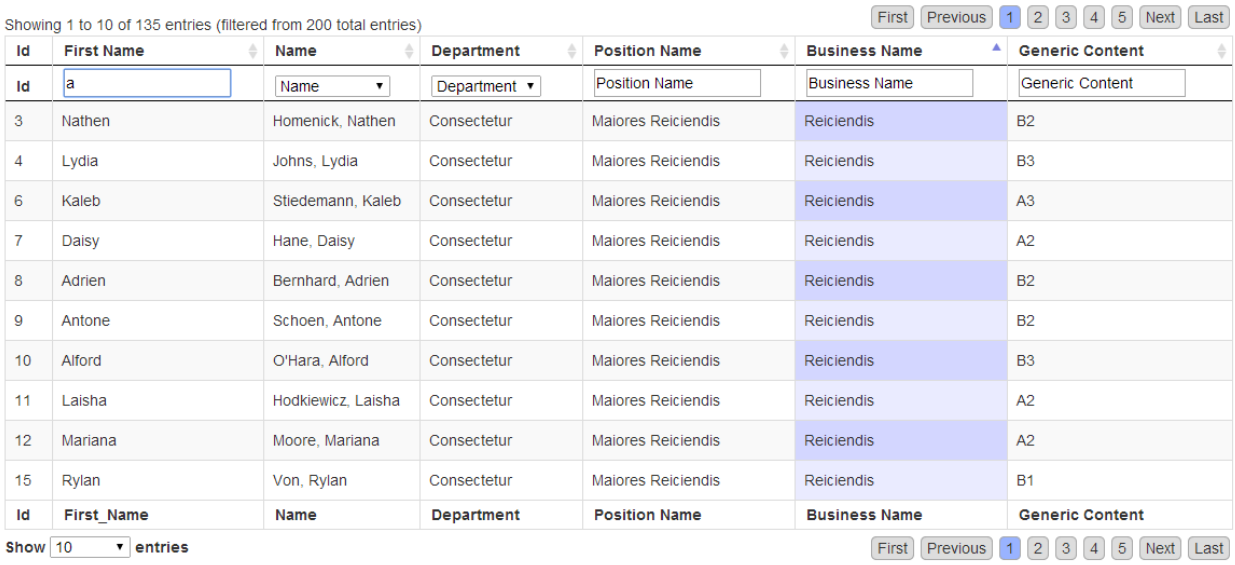
```
class Staff(models.Model):

    first_name = models.CharField(max_length=255, help_text=_("Enter the name of the␣
↪staff being rounded"))
    last_name = models.CharField(max_length=255, help_text=_("Enter the name of the␣
↪staff being rounded"))
    active = models.CharField(max_length=10, choices = ACTIVE_CHOICES)

    position = models.ForeignKey(Position)
    department = models.ForeignKey(Department)

    limit = models.Q(app_label='staff', model='genericmodela') | models.Q(app_label=
↪'staff', model='genericmodelb')
    content_type = models.ForeignKey(ContentType, limit_choices_to=limit)
    object_id = models.PositiveIntegerField()
    generic_object = generic.GenericForeignKey("content_type", "object_id")
```

to a filterable/orderable table in a template like this with as little code as possible:



Showing 1 to 10 of 135 entries (filtered from 200 total entries)

First Previous 1 2 3 4 5 Next Last

Id	First Name	Name	Department	Position Name	Business Name	Generic Content
Id	a	Name	Department	Position Name	Business Name	Generic Content
3	Nathen	Homenick, Nathen	Consectetur	Maiores Reiciendis	Reiciendis	B2
4	Lydia	Johns, Lydia	Consectetur	Maiores Reiciendis	Reiciendis	B3
6	Kaleb	Stiedemann, Kaleb	Consectetur	Maiores Reiciendis	Reiciendis	A3
7	Daisy	Hane, Daisy	Consectetur	Maiores Reiciendis	Reiciendis	A2
8	Adrien	Bernhard, Adrien	Consectetur	Maiores Reiciendis	Reiciendis	B2
9	Antone	Schoen, Antone	Consectetur	Maiores Reiciendis	Reiciendis	B2
10	Alford	O'Hara, Alford	Consectetur	Maiores Reiciendis	Reiciendis	B3
11	Laisha	Hodkiewicz, Laisha	Consectetur	Maiores Reiciendis	Reiciendis	A2
12	Mariana	Moore, Mariana	Consectetur	Maiores Reiciendis	Reiciendis	A2
15	Rylan	Von, Rylan	Consectetur	Maiores Reiciendis	Reiciendis	B1
Id	First_Name	Name	Department	Position Name	Business Name	Generic Content

Show 10 entries

First Previous 1 2 3 4 5 Next Last

There are a couple of other similar projects worth checking out to see if they fit your needs better:

- django-datatables-view
- django-datatables
- django-eztables

CHAPTER 6

Installation

```
$ pip install django-listable
```

CHAPTER 7

Settings

Listable currently has 4 settings you can configure to be used as default values for your table (they can be overriden in the listable template tag).

LISTABLE_DOM

Default datatables sDOM parameter to use. By default listable uses the Bootstrap 3 dom below.:

```
# bootstrap 2
# LISTABLE_DOM = '<"row-fluid"<"span6"ir><"span6"p>>rt<"row-fluid"<"span12"lp>>'

#boostrap 3
LISTABLE_DOM =  '<"row"<"col-sm-6"i><"col-sm-6"rp>>rt<"row"<"col-sm-12"lp>>'
```

LISTABLE_PAGINATION_TYPE

```
# pagination types -> bootstrap2, bootstrap3, two_button, full_numbers
LISTABLE_PAGINATION_TYPE = "full_numbers"
```

LISTABLE_STATE_SAVE

Enable sticky filters by default.:

```
LISTABLE_STATE_SAVE = True
```

LISTABLE_PAGINATE_BY

Default page size.:

```
LISTABLE_PAGINATE_BY = 10
```

CHAPTER 8

Usage

There's four steps to using django-listable

1. Including *listable* in your settings.INSTALLED_APPS
2. Create a view by subclassing listable.views.BaseListableView
3. Connect the view to a url pattern in your apps urls.py
4. Include the *listable* template tag in a template

These steps will demonstrated below assuming we have a Django application called staff and we want to create a page on our site with a list of staff and the department and business they belong to.

with the following models defined:

```
class Business(models.Model):

    name = models.CharField(max_length=255)


class Department(models.Model):

    name = models.CharField(max_length=255)
    business = models.ForeignKey(Business)


class Staff(models.Model):

    first_name = models.CharField(max_length=255, help_text=_("Enter the name of the␣
↪staff being rounded"))
    last_name = models.CharField(max_length=255, help_text=_("Enter the name of the␣
↪staff being rounded"))
    active = models.CharField(max_length=10, choices = ACTIVE_CHOICES)

    department = models.ForeignKey(Department)

    def name(self):
```

```
        return "%s, %s" % (self.last_name, self.first_name)

    def status(self):
        return self.get_active_display()
```

A full functional example can be found in the demo app included with django-listable.

Adding *listable* to settings.INSTALLED_APPS

To start using django-listable add *listable* to your INSTALLED_APPS:

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.admin',


    'staff',
    'listable',
    ...
)
```

Defining a Listable view

To define a *listable* view, sublcass *listable.views.BaseListableView* and set the model that is to be used as the source of data:

```
from listable.views import BaseListableView
from models import Staff


class StaffList(BaseListableView):

    model = models.Staff

    ...
```

Defining Columns for your table

Every *listable* view must define one or more fields to be displayed as columns in the table. *listable* fields are defined in a manner similar to ModelForms:

```
class StaffList(BaseListableView):

    model = models.Staff
```

```
    fields = (...)
    widgets = {...} # optional
    search_fields = {...} # optional
    order_fields = {...} # optional
    headers = {...} # optional
    select_related = (...) # optional
    prefetch_related = (...) # optional
    order_by = (..) # optional
```

fields

Fields defines an iterable of the columns that you want to display in the table, these fields can either be fields on your model, foreign key lookups, the name of a callable on your view, the name of a callable on your model or the result of an *extra* query.

widgets

Widgets is a dictionary mapping a field to a search widget type. Currently you can use either text (default) or select inputs. For example:

```
from listable.views import BaseListableView, SELECT

from . import models

class StaffList(BaseListableView):

    model = models.Staff

    fields = ("id", "name", "active", "department__name",)

    widgets = {
        "department__name": SELECT,
        "active": SELECT,
    }
```

The choices available in a select widget are currently automatically populated although this will change to allow manual configuration of choices in the future. The choices are populated based on either the *choices* option for a model field or in the case of a foreign key all the values of the foreign key lookup. (*I hope to make this more flexible in the future*)

search_fields (optional)

Search fields are a mapping of field names to the django filter syntax that should be used for searching the table. This can either be a string, an iterable of strings or a falsy value to disable searching on that field. For example:

```
search_fields = {
    "name": ("first_name__icontains", "last_name__icontains",),
    "last_name": "last_name__exact",
    "genericname": "genericname__icontains",
    "department__name": False,
}
```

if a field is not declared in search_field's it a filter using *icontains* is assumed.

order_fields (optional)

Order fields allows you to define how a column should be ordered (similar to Django's ordering or order_by). For example:

```
order_fields = {
    "name": ("last_name", "first_name",),
}
```

headers (optional)

Headers is a mapping of field names to the column name to be displayed. For example by default a field name of *department__business__name* would be converted to "Department Business Name" but that could be overriden like so:

```
headers = {
    "department__business__name": _("Business"),
}
```

select_related

Allows you to use Django's queryset select_related option for reducing database queries. e.g:

```
select_related = ("department", "position", "department__business",)
```

prefetch_related

Allows you to use Django's queryset prefetch_related option for reducing database queries. e.g:

```
prefetch_related = ("some_fk__some_field",)
```

get_extra

Due to a bug with pagination, using an extra query will result in your entire table being loaded into memory before being paginated :(

You may define a callable *get_extra* method on your view that should return a dictionary suitable for use in the Django queryset's *extra* method. For example:

```
def get_extra(self):
    return {select: {'is_recent': "pub_date > '2006-01-01'"}}
```

A more complex example is given in the "Complete Example" sample below.

order_by

Allows you to set the default ordering for a view:

```
order_by = ("position", "name",,)
```

Formatting fields

The order in which *listable* tries to find a method for formatting a field for display is as follows:

1. A method on the actual view:

```
class StaffList(BaseListableView):

    model = models.Staff

    fields = (..., "name",...)
    def name(self, staff):
        return staff.name()
```

2. A *get_{field}_display* callable on the model.
3. A callable on the model:

```
class Staff(Model):
    ...
    def staff_name(self):
        return "{0} {1}".format(self.first_name, self.last_name)

class StaffList(BaseListableView):

    model = models.Staff

    fields = (..., "staff_name",...)
```

4. A field on the model.

A *listable* column is defined using the *listable.views.Column* data structure. A *Column* is essentially a namedtuple with the following fields (detailed descriptions below):

Including the *listable* template tag in a template

To include *listable* in your templates you need to load the *listable* template tags and include the *listable_css*, a placeholder for the listable table and the listable tag which tells the template the name of the view to wire the table to.:

```
{% extends 'base.html' %}

{% load listable %}

{% block extra_css %}
    {% listable_css %}
{% endblock extra_css %}

{% block content %}
    {{listable_table}}
{% endblock %}

{% block extra_js %}
{% listable 'staff-list'%}
{% endblock extra_js %}
```

with the example above requiring a url something like:

```
urlpatterns = patterns('',
    url('staff-list/$', views.StaffList.as_view(), name="staff-list"),
)
```

Arguments to the listable tag

The listable tag currently has 1 required argument and five optional keyword args. A full example of the listable template tag looks like:

```
{% listable 'staff-list' dom="", save_state=False, pagination_type="", css_table_
↪class="", css_input_class="" %}
```

dom

Overrides the default Datatables sDOM parameter to use.

```
{% listable 'staff-list' dom='<"row-fluid"<"span6"ir><"span6"p>>rt<"row-fluid"<"span12
↪"lp>>' %}
```

pagination_type

Overrides the default Datatables sDOM parameter to use.

```
{% listable 'staff-list' pagination_type='bootstrap3' %}
```

save_state

Save state enables/disables sticky filters in DataTables.:

```
{% listable 'staff-list' save_state=False %}
```

css_table_class

Add a css class to your datatables table e.g.:

```
{% listable 'staff-list' css_table_class="striped compact" %}
```

css_input_class

Add a css class to the datatables column filter inputs e.g.:

```
{% listable 'staff-list' css_table_class="input-sm" %}
```

CHAPTER 9

A Complete Example

This is a complete example of a *django-listable* table. It is included as a demo app under the django-listable/listable-demo/

models.py

```
ACTIVE = 'active'
INACTIVE = 'inactive'
TERMINATED = 'terminated'

ACTIVE_CHOICES = (
    (ACTIVE, "Active"),
    (INACTIVE, "Inactive"),
    (TERMINATED, "Terminated"),
)

ACTIVE_CHOICES_DISPLAY = dict(ACTIVE_CHOICES)


class Business(models.Model):

    name = models.CharField(max_length=255)
    business_type = models.IntegerField(choices=zip(range(5), range(5)), default=1)

    class Meta:
        verbose_name_plural = "Businesses"

    def __unicode__(self):
        return self.name


class Department(models.Model):
```

```
    name = models.CharField(max_length=255)
    business = models.ForeignKey(Business)

    def __unicode__(self):
        return self.name


class Position(models.Model):

    name = models.CharField(max_length=255)

    def __unicode__(self):
        return self.name


class AbstractGeneric(models.Model):

    name = models.CharField(max_length=255)
    description = models.TextField()

    staff = generic.GenericRelation(
        "Staff",
        content_type_field="content_type",
        object_id_field="object_id",
    )

    class Meta:
        abstract = True


class GenericModelA(AbstractGeneric):

    class Meta:
        verbose_name_plural = "Generic Model A's"

    def __unicode__(self):
        return self.name


class GenericModelB(AbstractGeneric):

    class Meta:
        verbose_name_plural = "Generic Model B's"

    def __unicode__(self):
        return self.name


class Staff(models.Model):

    first_name = models.CharField(max_length=255, help_text=_("Enter the name of the␣
↪staff being rounded"))
    last_name = models.CharField(max_length=255, help_text=_("Enter the name of the␣
↪staff being rounded"))
    active = models.CharField(max_length=10, choices=ACTIVE_CHOICES)

    position = models.ForeignKey(Position)
    department = models.ForeignKey(Department)
```

```
    limit = models.Q(app_label='staff', model='genericmodela') | models.Q(app_label=
↪'staff', model='genericmodelb')
    content_type = models.ForeignKey(ContentType, limit_choices_to=limit)
    object_id = models.PositiveIntegerField()
    generic_object = generic.GenericForeignKey("content_type", "object_id")

    class Meta:
        verbose_name_plural = "staff"
        ordering = ("last_name", "first_name",)

    def name(self):
        return "%s, %s" % (self.last_name, self.first_name)

    def status(self):
        return ACTIVE_CHOICES_DISPLAY[self.active]

    def __unicode__(self):
        return self.name()
```

views.py

```
class StaffList(BaseListableView):

    model = models.Staff

    fields = (
        "id",
        "name",
        "active",
        "department__name",
        "position__name",
        "department__business__name",
        "department__business__business_type",
        "genericname",
    )

    widgets = {
        "department__business__name": SELECT,
        "department__business__business_type": SELECT,
        "position__name": SELECT,
        "choices": SELECT,
        "active": SELECT,
    }

    search_fields = {
        "name": ("first_name__icontains", "last_name__icontains",),
        "last_name": "last_name__exact",
        "genericname": "genericname__icontains",
        "department__name": "department__name__icontains",
    }

    order_fields = {
        "name": ("last_name", "first_name",),
    }
```

```
    headers = {
        "position__name": _("Position"),
        "department__business__name": _("Business"),
        "department__business__business_type": _("Business Type"),
    }

    select_related = ("department", "position", "department__business",)

    def generic(self, obj):
        return obj.generic_object.name

    def name(self, staff):
        return staff.name()

    def get_extra(self):
        cta = ContentType.objects.get_for_model(models.GenericModelA)
        ctb = ContentType.objects.get_for_model(models.GenericModelB)

        extraq = """
        CASE
            WHEN content_type_id = {0}
                THEN (SELECT name from staff_genericmodela WHERE object_id = staff_
↪genericmodela.id)
            WHEN content_type_id = {1}
                THEN (SELECT name from staff_genericmodelb WHERE object_id = staff_
↪genericmodelb.id)
        END
        """.format(cta.pk, ctb.pk)

        return {"select": {'genericname': extraq}}
```

staff_list.html

```
{% extends 'base.html' %}

{% load listable %}

{% block extra_css %}
    {% listable_css %}
{% endblock extra_css %}

{% block content %}
    {{listable_table}}
{% endblock %}

{% block extra_js %}
{% listable 'staff-list' save_state=True %}
{% endblock extra_js %}
```