Django Slick Reporting

Release 0.5.8

Apr 08, 2021

Contents:

Django Slick Reporting a report engine allowing you to create & display diverse analytics. Batteries like a ready to use View and Highcharts & Charts.js integration are included.

CHAPTER 1

Installation

To install django-slick-reporting:

1. Install with pip: *pip install django-slick-reporting*.
2. Add `slick_reporting'` to `INSTALLED_APPS`.
3. For the shipped in View, add `'crispy_forms'` to `INSTALLED_APPS` and add `CRISPY_TEMPLATE_PACK = 'bootstrap4'` to your `settings.py`
4. Execute *python manage.py collectstatic* so the JS helpers are collected and served.

CHAPTER 2

Demo site

https://django-slick-reporting.com is a quick walk-though with live code examples

CHAPTER 3

Quickstart

You can start by using `SlickReportView` which is a subclass of `django.views.generic.FormView`

```
# in views.py
from slick_reporting.views import SlickReportView
from slick_reporting.fields import SlickReportField
from .models import MySalesItems

class MonthlyProductSales(SlickReportView):
    # The model where you have the data
    report_model = MySalesItems

    # the main date field used for the model.
    date_field = 'date_placed' # or 'order__date_placed'
    # this support traversing, like so
    # date_field = 'order__date_placed'

    # A foreign key to group calculation on
    group_by = 'product'

    # The columns you want to display
    columns = ['title',
               SlickReportField.create(method=Sum, field='value', name='value__sum',␣
↪verbose_name=_('Total sold $'))
               ]

    # Charts
    charts_settings = [
     {
        'type': 'bar',
        'data_source': 'value__sum',
        'title_source': 'title',
     },
    ]
```

Next step *Structure*

3.1 Structure

If you haven't, please check https://django-slick-reporting.com for a quick walk-though with live code examples..

And now, Let's explore the main components of Django Slick Reporting and what setting you can set on project level.

3.1.1 Components

1. Result Field: represent a number, a calculation unit, for example: a Sum of a certain field. The report field identifies how the calculation should be done. ResultFields can depend on each other.
2. Generator: Represent a concrete report structure.If it would group by certain field, do a time series or a cross tab, and which columns (Report Field) to be calculated. It's also responsible for computing and provides low level access. *ie you can get the data in a list of dict/objects*
3. View: Responsible for creating a nice form to filter the report on, pass those filters to the generator class to create the report. It mimic the Generator API. Provide high level access. *You can hook it to your urls.py and you're all set, with the charts.*
4. Charting JS helpers: Django slick Reporting comes with highcharts and Charts js helpers libraries to plot the charts generated by the View

3.1.2 Settings

1. `SLICK_REPORTING_DEFAULT_START_DATE`: Default: the beginning of the current year
2. `SLICK_REPORTING_DEFAULT_END_DATE`: Default: the end of the current year.
3. `SLICK_REPORTING_FORM_MEDIA`: Controls the media files required by the search form. Defaults is:

```
SLICK_REPORTING_FORM_MEDIA = {
'css': {
    'all': (
        'https://cdn.datatables.net/v/bs4/dt-1.10.20/datatables.min.css',
        'https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.3/Chart.min.css',
    )
},
'js': (
    'https://code.jquery.com/jquery-3.3.1.slim.min.js',
    'https://cdn.datatables.net/v/bs4/dt-1.10.20/datatables.min.js',
    'https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.3/Chart.bundle.min.js',
    'https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.9.3/Chart.min.js',
    'https://code.highcharts.com/highcharts.js',
    )
}
```

4. `SLICK_REPORTING_DEFAULT_CHARTS_ENGINE`: Controls the default chart engine used.

3.2 The Slick Report View and form

3.2.1 What is SlickReportView?

SlickReportView is a CBV that inherits form `FromView` and expose the report generator needed attributes. it also

- Auto generate the search form

- return the results as a json response if ajax request
- Works on GET and POST

3.2.2 How the search form is generated ?

Behind the scene, Sample report calls `slick_reporting.form_factory.report_form_factory` a helper method which generates a form containing start date and end date, as well as all foreign keys on the report_model.

3.2.3 Override the Form

The Form purposes are

1. Provide the start_date and end_date
2. provide a `get_filters` method which return a tuple (Q_filers , kwargs filter) to be used in filtering. q_filter: can be none or a series of Django's Q queries kwargs_filter: None or a dict of filters

Following those 2 recommendation, your awesome custom form will work as you'd expect.

3.2.4 Charting

Charts settings is a list of objects which each object represent a chart configurations.

- type: what kind of chart it is: Possible options are bar, pie, line and others subject of the underlying charting engine. Hats off to : Charts.js.
- engine_name: String, default to `SLICK_REPORTING_DEFAULT_CHARTS_ENGINE`. Passed to front end in order to use the appropriate chart engine. By default supports *highcharts* & *chartsjs*.
- data_source: Field name containing the numbers we want to plot.
- title_source: Field name containing labels of the data_source
- title: the Chart title. Defaults to the *report_title*.
- plot_total if True the chart will plot the total of the columns. Useful with time series and crosstab reports.

On front end, for each chart needed we pass the whole response to the relevant chart helper function and it handles the rest.

3.2.5 The ajax response structure

Understanding how the response is structured is imperative in order to customize how the report is displayed on the front end.

Let's have a look

```
# Ajax response or `report_results` template context variable.
response = {
    # the report slug, defaults to the class name all lower
    "report_slug": "",

    # a list of objects representing the actual results of the report
    "data": [
```

(continues on next page)

(continued from previous page)

```
        {"name": "Product 0", "quantity__sum": "1774", "value__sum": "8758", "field_n
↪" : "value_n"},
        # .....
    ],

    # A list explaining the columns/keys in the data results.
    # ie: len(response.columns) == len(response.data[i].keys())
    # Contains needed information about the verbose name , if summable , hints about␣
↪the data type.
    "columns": [
        {"name": "name",
         "computation_field": "",
         "verbose_name": "Name",
         "visible": True,
         "type": "CharField",
         "is_summable": False
         },
        {"name": "quantity__sum",
         "computation_field": "",
         "verbose_name": "Quantities Sold",
         "visible": True,
         "type": "number",
         "is_summable": True},
        {"name": "value__sum",
         "computation_field": "",
         "verbose_name": "Value $",
         "visible": True,
         "type": "number",
         "is_summable": True}
    ],

    # Contains information about the report as whole if it's time series or a a␣
↪crosstab
    # And what's the actual and verbose names of the time series or crosstab specific␣
↪columns.
    "metadata": {"time_series_pattern": "",
                 "time_series_column_names": [],
                 "time_series_column_verbose_names": [],
                 "crosstab_model": '',
                 "crosstab_column_names": [],
                 "crosstab_column_verbose_names": []
                 },


    # a mirror of the set charts_settings on the SlickReportView
    # SlickReportView populates the id if missing and fill the `engine_name' if not␣
↪set
    "chart_settings": [
        {"type": "pie",
        'engine_name': 'highcharts',
         "data_source": ["quantity__sum"],
         "title_source": ["name"],
         "title": "Pie Chart (Quantities)",
         "id": "pie-0"},

        {"type": "bar",
        "engine_name": "chartsjs",
```

(continues on next page)

(continued from previous page)

```
        "data_source": ["value__sum"],
        "title_source": ["name"],
        "title": "Column Chart (Values)",
         "id": "bar-1"}
    ]
}
```

3.3 Report Generator API

The main class responsible generating the report and managing the flow

3.3.1 ReportGenerator

3.4 Computation Field API

Responsible for preforming the calculation.

3.4.1 ReportField Basic Structure:

Earlier in he docs you saw the computation fields `'__total__quantity__'` Let's see how it's written in *slick_reporting.fields*

```
from slick_reporting.fields import SlickReportField
from slick_reporting.decorators import report_field_register


@report_field_register
class TotalQTYReportField(SlickReportField):

    # The name to use when using this field in the generator
    name = '__total_quantity__'

    # the field we want to compute on
    calculation_field = 'quantity'

    # What method we want
    calculation_method = Sum  # the default

    # A verbose name
    verbose name = 'Total quantity'
```

If you want AVG to the field *price* then the ReportField would look like this

```
from django.db.models import Avg

@report_field_register
class TotalQTYReportField(SlickReportField):

    name = '__avg_price__'
    calculation_field = 'price'
```

(continues on next page)

(continued from previous page)

```
    calculation_method = Avg
    verbose name = 'Avg. Price'
```

3.4.2 How it works ?

The ReportGenerator is initialized with the needed configuration, it generates a list of the needed fields to be displayed and computed. For each computation field, it's given the filters needed and asked to get all the results prepared. **The preparation is a duty of the ReportField anyway**, then for each report_model record, the ReportGenerator again asks each ComputationField to get the data it has for each record and map it where it belongs.

3.4.3 Bundled Report Fields

- __total__ : Sum of the field names value
- __total_quantity__ :Sum of the field names 'quantity'
- __fb__ : Sum of the field value on the start date (or the start date of the active time series window)
- __balance__: Compound some of the field *value* .

Difference between total and balance is:

The field __total__ will return that client 1 bought 10 in Jan, 12 in Feb , 13 in March. while __balance__ will report client compound buy: 10 in Jan, 22 in Feb and 35 in March

3.4.4 Registering Report Field

To make this ReportField class available to the report, it has to be registered via `report_field_register`

Say you want to further customize your calculation, maybe you need to run a complex query

You can override both of those method and control the calculation

3.4.5 Calculation Flow:

ReportGenerator call

1. prepare
2. resolve

3.4.6 Two side calculation

todo: # Document how a single field can be computed like a debit and credit.

3.4.7 SlickReportField API

CHAPTER 4

Indices and tables

- genindex
- modindex
- search