DynamicForms Documentation

Release 0.9 dev

Velis Ltd

Apr 21, 2023

Contents:

CHAPTER 1

What is DynamicForms?

DynamicForms performs all the visualisation & data entry of your DRF Serializers & ViewSets and adds some candy of its own: It is a django library that gives you the power of dynamically-shown form fields, auto-filled default values, dynamic record loading and similar candy with little effort. To put it differently: once defined, a particular ViewSet / Serializer can be rendered in multiple ways allowing you to perform viewing and authoring operations on the data in question.

It is based on django-rest-framework

1.1 Why DynamicForms

- Turn your rest-framework ViewSets into HTML forms
- Powerful HTML based CRUD
  - Support for fetching "new" records, both in JSON or in HTML
  - Render to HTML, dialog html or from your own template
  - Render form (embedded or dialog) or table, depending on situation
  - Easily add actions and place the buttons to execute them anywhere you like
- Clear separation of list & dialog templates
- Dynamic loading of additional records for table views
- Easy implementation of simple filtering
- Action items, declared globally, placed where you need them
- Custom templates whenever & wherever you want them
- Render to full html or work with dialogs within same page or both at the same time
- Each form and field have a unique HTML id for easy finding & manipulation
- Bootstrap 3 & 4 templates, jQuery UI coming soon, easy to make your own or enhance existing

- Support for form validation, will show errors even if they are not tied to a field
- Convenient JS functions for easier action scripting
- Progress dialog for long lasting ajax operations

CHAPTER 2

Using DynamicForms

2.1 Quick start guide

```
pip install dynamicforms
```

Then you need to add following setting to your project's settings.py .. code-block:: python

> **caption** settings.py
>
> **name** settings.py

REST_FRAMEWORK = {

> **'DEFAULT_RENDERER_CLASSES': (** 'rest_framework.renderers.JSONRenderer', 'rest_framework.renderers.BrowsableAPIRenderer', 'dynamicforms.renderers.TemplateHTMLRenderer',
>
>)

}

DynamicForms has been designed to cause minimal disruption to your existing code patterns.

So instead of DRF ModelViewSet just use DynamicForms ModelViewSet, instead of ModelSerializer - DynamicForms ModelSerializer.

Currently only the `ModelViewSet` is supported for ViewSets. We have others planned, but not implemented yet.

```
:caption: examples/rest/page_load.py
:name: examples/rest/page_load.py

from dynamicforms import serializers, viewsets
from ..models import PageLoad


class PageLoadSerializer(serializers.ModelSerializer):
    form_titles = {
```

(continues on next page)

(continued from previous page)

```
        'table': 'Dynamic page loader list',
        'new': 'New object',
        'edit': 'Editing object',
    }

    class Meta:
        model = PageLoad
        exclude = ()


class PageLoadViewSet(viewsets.ModelViewSet):
    template_context = dict(url_reverse='page-load')
    pagination_class = viewsets.ModelViewSet.generate_paged_loader(30)  # enables␣
↪pagination

    queryset = PageLoad.objects.all()
    serializer_class = PageLoadSerializer
```

Listing 1: examples/models.py (excerpt)

```
from django.db import models

class PageLoad(models.Model):
    """
    Shows how DynamicForms handles dynamic loading of many records in ViewSet result
    """
    description = models.CharField(max_length=20, help_text='Item description')
```

Filter will be applied if user press enter in filter field. If you want to have filter button in list header, call Actions with add_default_filter = True.

Listing 2: examples/filter.py

```
from dynamicforms import serializers, viewsets
from dynamicforms.action import Actions
from ..models import Filter


class FilterSerializer(serializers.ModelSerializer):
    form_titles = {
        'table': 'Dynamic filter list',
        'new': 'New object',
        'edit': 'Editing object',
    }
    actions = Actions(add_default_crud=True, add_default_filter=True)

    class Meta:
        model = Filter
        exclude = ()


class FilterViewSet(viewsets.ModelViewSet):
    template_context = dict(url_reverse='filter')
    pagination_class = viewsets.ModelViewSet.generate_paged_loader(30)  # enables␣
↪pagination
```

(continues on next page)

(continued from previous page)

```
    queryset = Filter.objects.all()
    serializer_class = FilterSerializer
```

2.1.1 Custom page template

Following is an example page template to render straight router URLs. Customise this to match your site's look & feel. The emphasized lines show the lines that obtain and render the actual data, be it table or form. See *DYNAMICFORMS_PAGE_TEMPLATE*.

Listing 3: examples/templates/examples/page.html

```
{% extends 'examples/base.html' %}
{% load dynamicforms %}
{% block title %}
  {{ serializer.page_title }}
{% endblock %}
{% block body %}
  {% get_data_template as data_template %}

<div class="{{ DYNAMICFORMS.bs_card_class }}">
  <div class="{{ DYNAMICFORMS.bs_card_header }}">
    {{ serializer.page_title }}
    {% if serializer.render_type == 'table' %}{% render_table_commands serializer
↪'header' %}{% endif %}
  </div>
  <div class="{{ DYNAMICFORMS.bs_card_body }}">
    {% include data_template with serializer=serializer data=data %}
  </div>
</div>
{% endblock %}
```

Done. Point your DRF router to the ViewSet you just created and your browser to its URL - make sure you add ".html" to the URL to specify the renderer. If you forget that, you will get DRF's API renderer.

2.2 Templates

2.2.1 Design

Templates are organised in template packs for different UI libraries. DynamicForms provides template packs for bootstrap v3 / v4 with jQuery UI templates pending.

```
DYNAMICFORMS_TEMPLATE = 'dynamicforms/bootstrap'
```

Main template is base.html. HTML and its head tag are defined here. There are three blocks, that can be used in deriving templates:

- title: For defining the title of HTML.
- head_extra: to add additional definitions or includes in head tag.
- body: to insert the body of HTML.

Head tag includes base_includes.html (for bootstrap we have base_includes_v3.html and base_includes_v4.html). Here all the libraries that are needed for dynamic forms to work are included.

Base_list.html can be used for rendering Viewset in list mode. It shows all records with values or »No data« label if there is no data. When user clicks on record (only if default CRUD functionlity is enabled), this record is shown in form (modal dialog or separate page) and can be edited there.

Base_form.html can be used for rendering ViewSet in form mode. It shows one record, and if crud is enabled in Viewset, it can also be edited.

Form can be shown as modal dialog. For that template which is defined in settings.py - modal_dialog_rest_template is used. When using bootstrap v4 default template is modal_dialog_v4.html.

Template for dialog should have first div with »dynamicforms-dialog« class. JS searches for that to see if the response from server was a dialog or other error message.

For showing fields base template the one that is defined in settings.py -field_base_template. For bootstrap v4 default template is field/base_field_v4.html. That template makes sure that the label, input, errors and help text is correctly shown. This template is extracted by templates that are used for rendering individual field types (e.g.: checkbox.html, input.html, radio.html, etc.)

Some additional functionalities are not supported with jQuery templates (e.g. copy to clipboard on Select 2 fields).

Todo: explain how to change template to another one

Todo: Custom page template

Todo: Custom dialog classes for dialog formatting

Todo: SingleRecordViewSet

Todo: RenderMixin - parameters and what they do

CHAPTER 3

DynamicForms Configuration

Like much of django-based libraries, DynamicForms is also configured from settings.py.

3.1 Activate DynamicForms in DRF

In order to activate DynamicFroms, you need to add its renderers to DRF configuration, like so:

```
REST_FRAMEWORK = {
    'DEFAULT_RENDERER_CLASSES': (
        'rest_framework.renderers.JSONRenderer',
        'rest_framework.renderers.BrowsableAPIRenderer',
        'dynamicforms.renderers.TemplateHTMLRenderer',
    )
}
```

DRF will remain in control of JSON & Browseable API renderers while we activate DynamicForms for *.html* renders.

Note: The DRF renderers are taken from default DRF configuration. If it should change, feel free to change the setting as well.

3.2 List of settings

DYNAMICFORMS_TEMPLATE
: Specifies the template pack that dynamicforms will use for rendering HTML forms, e.g. 'bootstrap', 'jQuery UI', etc.

DYNAMICFORMS_PAGE_TEMPLATE
: Specifies the main page template to be used for plain rendering when you navigate to a ViewSet router URL.

Defaults to DYNAMICFORMS_TEMPLATE + 'page.html' (but currently, there's nothing there - see dynamicforms examples on how to specify base page template)

`DYNAMICFORMS_TEMPLATE_OPTIONS`

Offers a chance to do some things in the template pack differently. It can be used for anything from choosing version of the underlying framework (bootstrap 3 vs 4) or rendering various subsections differently (e.g. horizontally aligned form labels vs vertically aligned ones or editing record in modal dialog vs editing in new page).

Supported bootstrap versions are v3 and v4.

`DYNAMICFORMS_MODAL_DIALOG`

Name of template for modal dialog. It will be appended any version modifiers, i.e. bootstrap version postfix if bootstrap template pack.

CHAPTER 4

DynamicForms API reference

4.1 Action controls

4.1.1 Class reference

class FormButtonAction(*btn_type: dynamicforms.action.FormButtonTypes*, *label: str = None*, *btn_classes: str = None*, *button_is_primary: bool = None*, *position: dynamicforms.action.FormPosition = <FormPosition.FORM_FOOTER: 2>*, *name: Optional[str] = None*, *serializer: rest_framework.serializers.Serializer = None*, *icon: Optional[str] = None*, *action=None*, *display_style=None*)

to_component_params(*row_data*, *serializer*)

generates a dict with parameters for component that is going to represent this action. none means don't render / activate this action on this row

Parameters

- **row_data** –
- **serializer** –

Returns

class FormButtonTypes

An enumeration.

class FormPosition

An enumeration.

class RenderableActionMixin(*label: str*, *title: str*, *icon: str = None*, *btn_classes: Union[str, dict, None] = None*, *display_style: Optional[dict] = None*)

Action that is rendered on screen

class TableAction(*position: dynamicforms.action.TablePosition*, *label: str*, *title: Optional[str] = None*, *icon: Optional[str] = None*, *field_name: Optional[str] = None*, *name: Optional[str] = None*, *serializer: rest_framework.serializers.Serializer = None*, *btn_classes: Union[str, dict, None] = None*, *action=None*, *display_style=None*)

to_component_params (*row_data*, *serializer*)
generates a dict with parameters for component that is going to represent this action. none means don't render / activate this action on this row

Parameters

- **row_data** –
- **serializer** –

Returns

class TablePosition
An enumeration.

4.2 Context processors

4.3 Dialogs

Dialogs offer more flexibility by allowing you to place them on screen when needed instead of having them pre-rendered in the page. Their contents are always fresh and theirHTML adapts to the task at hand. For example, when user makes a mistake entering data, the returned dialog will already contain all the warnings in the HTML.

To use, either add *?df_render_type=dialog* to URL or add a *X_DF_RENDER_TYPE=dialog* HTTP header to request.

The default dialog templates allow for some customisation of dialog rendered.

4.3.1 Dialog classes

```
class MyViewSet(viewsets.ModelViewSet):
    template_context = dict(url_reverse='my-item', dialog_classes='dialog-lg')
```

Good for specifying additional dialog classes, like how large the dialog should be.

4.3.2 Dialog header classes

```
class MyViewSet(viewsets.ModelViewSet):
    template_context = dict(url_reverse='my-item', dialog_header_classes='bg-info')
```

Good for specifying additional dialog header classes, like typeof dialog (warning, info, primary, etc)

4.4 Fields

Re-declares all DRF fields such that they also inherit DynamicForms' mixins.

To use, import like so:

```
from dynamicforms.fields import {your desired field classes}
```

Make sure you don't import DRF's field classes over these.

4.4.1 Class reference

4.4.2 Field mixins

4.5 Renderers

Contains template renderers.

To use, declare renderers in settings.py, like so:

```
REST_FRAMEWORK = {
    'DEFAULT_RENDERER_CLASSES': (
        'rest_framework.renderers.JSONRenderer',
        'rest_framework.renderers.BrowsableAPIRenderer',
        'dynamicforms.renderers.TemplateHTMLRenderer',
    )
}
```

4.5.1 Class reference

4.6 Serializers

Currently only provides ModelSerializer for model-based data manipulation.

To use, import like so:

```
from dynamicforms.serializers import ModelSerializer
```

Make sure you don't import DRF's ModelSerializer over this one.

4.6.1 Class reference

4.7 Template tags

4.7.1 Tag reference

4.7.2 Filter reference

4.8 ViewSets

Currently only provides ModelViewSet for model-based data manipulation.

To use, import like so:

```
from dynamicforms.viewsets import ModelViewSet
```

Make sure you don't import DRF's ModelViewSet over this one.

4.8.1 Class reference

CHAPTER 5

Contributor's documentation

5.1 Code style guide

5.1.1 Python

Rules are made to be broken, so…

PEP 8 with the following exceptions:

- E5 - line length: use 120 character limit
- E722 - (bare excepts) too broad
- E731 - do not assign a lambda expression, use a def

Our rationale for breaking E722 & E731:

- E722 - sometimes you just need code that doesn't except. Better to do nothing than to break. Depending on importance, such except blocks might have some sort of report added so that the programmer knows something happened.
- E731 - when you have tons of tiny declarations, using a lambda means 1 line of code while using def means 3 lines or more

Note that E722 & E731 may not be broken at all in this project. It's just that our linters have these checks disabled on principle. Apologies… but no regrets.

5.1.2 JavaScript

This one is a lot tougher. It is a work in progress, but I hope we (internally) sort it out before you (the would-be contributor) ever see this.

While we would **love** to adopt one of the popular style guides, we so far haven't managed to. No time doing research and finding a style that doesn't break what we feel is important. Suggestions based on the following welcome:

- Pure EcmaScript 5.1

    - No code conversion tools: the code in the repository runs straight out of the box

- Semicolons mandatory
- 2 spaces indentation
- use == instead of ===: type coercion is helpful

The above straight out eliminates:

- standard

These remain in play (we found all 5 on some "most popular linters" list)

- Airbnb
- Google
- Idiomatic
- jQuery

Note: While we're seriously considering EcmaScript 6, so far we haven't really encountered a situation where it would be absolutely required to do something. We try to be as light as possible on JavaScript anyway and while using EcmaScript 5.1 may mean a couple of lines more code, it also keeps away transpilers & missing-feature-completion scripts.

5.1.3 Documentation

- 120 character line length limit

5.2 dynamicforms_dev package

This package contains a helpful code generator so that we don't have to copy-paste an interface change to 35-ish different fields every time we add a cool new feature.

5.2.1 Generate fields

The helpful code generator. :)

This command is used for generating *fields/__init__.py*.

1. First it finds all the field types from *rest_framework/fields.py*
2. Then take care of all the imports that are needed for *fields/__init__.py*.
3. Then it finds all the parameters, that can be used to set up field and includes them in *__init__* functions for individual field. This we have done so that code completion might work better in your IDE. In the end it adds another parameter **kw which makes sure that any new parameters added in newer versions of DRF don't break the functionality.

```
python manage.py generate_fields
```

5.3 Requirements for running tests

```
pip install selenium
```

5.3.1 Gecko driver

Geckodriver is needed for the functional tests. It is available from https://github.com/mozilla/geckodriver/releases. You need to download and extract it and put it somewhere on your system path.

For macOS or Linux, one convenient place to put it is ~/.local/bin

For Windows, put it in your Python Scripts folder

To test that you've got this working, open up a Bash console and you should be able to run:

```
geckodriver --version
geckodriver 0.17.0
```

The source code of this program is available at https://github.com/mozilla/geckodriver.

This program is subject to the terms of the Mozilla Public License 2.0.

You can obtain a copy of the license at https://mozilla.org/MPL/2.0/.

5.4 creating your own template pack

5.4.1 Context variables referring to template packs

These variables will be set in a `dynamicforms.viewsets.ModelViewSet` class declaration.

`crud_form: True | False`
If set, template pack is expected to render HTML that will allow user to edit the presented data.

Todo: The above is left just to keep an example. Currently there is nothing in the code that would be used

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index