Dynamic forms

# List of features

* 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
* Easy separation of list & dialog templates
* 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

# Explaining Setings.py

## MODULE\_PREFIX

Module prefix is DYNAMICFORMS\_

## TEMPLATE

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

## 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.

Following settings are calculated constants specifying the HTML header includes providing js and css for desired Bootstrap version.

## MODAL\_DIALOG

Name of template for modal dialog.

## BSERVER\_INCLUDES

Path to template for includes for bootstrap version.

## BSVER\_FIELD\_TEMPLATE

Path to base template for fields.

## BSVER\_MODAL

Path to template for modal dialog

## CONTEXT\_VARS

All settings from settings.py can be accessed in all form renderers by CONTEXT\_VARS

# How to create standard ViewSet

Instead of DRF ModelViewSet use Dynamic forms ModelViewSet. It works just like DRF's, but it has some extra features:

* Two templates can be used.
  + Variable *list\_template\_name* for list view
  + Variable *template\_name* for form view
* In template\_context variable one could set if this is a CRUD form or not
* While adding records, fields can have predefined values.

To render viewset as API or JSON use the same way as in DFR. To render it in HTML you can just replace last slash (»/«) with ».html« at the end of URL.

To render viewset for new record use pk=new. To render viewset for editing record use pk={record\_id}

# Mixins

UUIDMixIn

Is used in fields and serializers, so every field and serializer gets its unique id.

In form where serializer is used, id is serializers uuid. Table with list of records has id »list-serializer.uuid«, in dialog id is »dialog-{serializer.uuid}« and save button's id on dialog is »save-{serializer.uuid}«

Similar is for fields. All inputs in HTML get id from field.uuid. Div that contains all that belongs to the field has »container-{field.uuid}« for id, label has »label-{field.uuid}« and help text (if exists) has »help-{field.uuid}« for id.

## NewMixIn

Is used in ModelViewSet. It allows us to use »new« for record's pk and then renders dialog, where new record can be inserted.

# Viewset rendering – list vs form

Viewsets could be rendered as list or as form. Template for list should be declared in *list\_template\_name* in Viewset declaration and template for form should be declared in *template\_name*. If there is no *list\_template\_name* declared, then template in *template\_name* will be used also for rendering list.

Program decides when to render list or form based on pk of record. If there is no pk given, then it will render list, otherwise it will render form.

# DF context variable

All settings from settings.py are accessible in templates through DF variable. Example for supporting different versions of bootstrap:

{% if DF.TEMPLATE\_VARIANT.BOOTSTRAP\_VERSION == 'v3' %}  
 {% set\_var card\_class='panel panel-default' card\_header='panel-heading' card\_body='panel\_body' %}  
{% else %}  
 {% set\_var card\_class='card' card\_header='card-header' card\_body='card-body' %}  
{% endif %}

# Template tags

## render\_form(serializer, template\_pack=None, form\_template=None)

Renders form from serializer. If form\_template is given, then renderer will use that one, otherwise it will use what is defined in self.base\_template (e.g.: »form.html«) from template\_pack (e.g.: »dynamicforms/bootstrap/field/«)

{% set\_var template\_pack=DF.TEMPLATE|add:'field' %}  
{% render\_form serializer template\_pack=template\_pack %}

## render\_field(field, sytle)

Renders separate field. Style is a dict, that contains template\_pack or form\_tempate and rendered. It is defined when rendering form, so it is best to use that one. To do that dynamicforms should be loaded first. See example below.

{% load dynamicforms %}

{% for field in form %}  
 {% if not field.read\_only %}  
 {% render\_field field style=style %}  
 {% endif %}  
{% endfor %}

## set\_var(context, \*\*kwds)

## Sets the given variables to provided values. Kind of like the 'with' block, only it isn't a block tag

## :param context: template context (automatically provided by django)

## :param kwds: named parameters with their respective values

## :return: this tag doesn't render

## set\_var\_conditional(context, condition=None, else\_value=None, \*\*kwds)

## Sets the given variables to provided values. Kind of like the 'with' block, only it isn't a block tag

## :param context: template context (automatically provided by django)

## :param kwds: named parameters with their respective values

## :param condition: a value which specifies the original assignment if truthy or else\_value if false

## :param else\_value: value to be assigned to the variable(s) when condition is false

## :return: this tag doesn't render

# Templates

## Design

Templates are organised in template packs for different UI libraries. Dynamicforms are providing template packs for bootstrap v3 and v4 (templates/dynamicforms/bootstrap) and for jQuery (templates/dynamicforms/jquery).

Main template is base.html. Here HTML and its head tag are defined. There are three blocks, that can be used in other 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 are included all the libraries that are needed for dynamic forms to work.

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, this record is shown in form (modal dialog or separate page) and can be edited there.

Base\_form.html can be used for rendering Viewsetin form mode. It shows one record, and if crud is enabled (*template\_context = dict(crud\_form=True)* in Viewset) it can also be edited.

Form can be shown as modal dialog. For that template which is defined in settings.py🡪BSVER\_MODAL 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🡪BSVER\_FIELD\_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.)

# Dynamicforms\_dev generate\_fields command

This command is used for generating dynamicforms.fields.\_\_init\_\_.py.

At first it finds all the field types from rest\_framework.fields.py

Then take care of all the imports that are needed for fields.\_\_init\_\_.py.

Then it finds all the parameters, that can be used to set up field in includes them in \_\_init\_\_ functions for individual field. This is necessary for IDE, so it can help with code completion. In the end it adds another parameter \*\*kw which is additional.

All of this is eventually printed into dynamicforms.fields.\_\_init\_\_.py