Django Bootstrap 4 Modals Documentation

Release 2.1.0

Christopher Trudeau

Nov 06, 2020

Contents

1 Installation **3**

2 Supports **5**

3 Docs & Source **7**

4 Methods **9**
4.1 Django Bootstrap Modals 4 9

5 Indices and tables **17**

Index **19**

A small library to help write Bootstrap 4 modals. It includes some base templates and some Javascript functions for quickly creating and displaying the dialogs.

CHAPTER 1

Installation

```
$ pip install django-bstrap-modals
```

Include `bsmodals` in your `INSTALLED_APPS` and make sure that your `APP_DIRS` setting inside of the `TEMPLATES` list is set to `True`.

CHAPTER 2

Supports

django-bstrap-modals has been tested with:

- Django 2.2.14 using Python 3.6
- Django 3.0.8 using Python 3.6

Due to the fact that it is a few templates and some Javascript, testing is currently manual, it likely works in other versions. Uses the new-style load static template tag, so may have problems prior to Django 2.1.

CHAPTER 3

Docs & Source

Docs: http://django-bstrap-modals.readthedocs.io/en/latest/

Source: https://github.com/cltrudeau/django-bstrap-modals

Version: 2.1.0

CHAPTER 4

Methods

4.1 Django Bootstrap Modals 4

This library wraps some Bootstrap 4 Modal dialog boxes as templates and provides some simple Javascript to help render them.

4.1.1 Setup

The library is broken into two parts, one for general dialogs and the other for a dialog that deals with REST API calls. Both sets require you to include the base library:

```
<script src="/static/bsmodals/bsmodals.js"></script>
```

If you want to use the REST API you also have to include:

```
<script src="/static/bsmodals/restapi.js"></script>
```

Additionally, dependingo on what dialogs you are using you will need to do a template `include` of either the supplied templates or your overwrittten version. See each dialog type below for specifics.

4.1.2 Built-in Dialogs

Alert Dialog

The alert dialog provides a simple pop-up with a button to clear it. It is used by including the `alert` template:

```
{% include "bsmodals/alert.html" %}
```

You should only include this once. Use the the Javascript helper function to show the dialog with different content.

bsmodals_alert (*title*, *msg*[, *style*])

Arguments

- **title** (*string*) – Dialog box title
- **msg** (*string*) – Message contained in the dialog box
- **style** (*string*) – Optional style parameter for the button, if not given, defaults to "btn-primary"

Example:

```
bsmodals_alert('Warning!', 'Will Robinson, you are in Danger!');
```

Error Dialog

The error dialog provides a simple pop-up with a button to clear it. It is used by including the `error` template:

```
{% include "bsmodals/error.html" %}
```

You should only include this once. Use the the Javascript helper function to show the dialog with different content.

bsmodals_error(*msg*[, *style*])

Arguments

- **msg** (*string*) – Message contained in the dialog box
- **style** (*string*) – Optional style parameter for the button, if not given, defaults to "btn-primary"

Example:

```
bsmodals_error('The sky is falling!', "btn-warning");
```

Confirm Dialog

The confirm dialog provides a pop-up with a "Yes" and "No" button. It should only be included once.

```
{% include "bsmodals/confirm.html" %}
```

You can use the Javascript helper function to show the dialog, changing the parameter on each call.

bsmodals_confirm(*title*, *msg*, *callback*[, *yes_text="Yes"*, *yes_style="btn-primary"*, *no_text="No"*, *no_style="btn-secondary"*])

Arguments

- **title** (*string*) – Title for the dialog
- **msg** (*string*) – Message contained in the dialog box
- **callback** – Callback function that takes a boolean, receives "true" if the user pressed "Yes" and "false" if they pressed "No"
- **yes_text** (*string*) – Optional text to use instead of "Yes" on the yes button
- **yes_style** (*string*) – Optional style for the yes button, defaults to "btn-primary"
- **no_text** (*string*) – Optional text to use instead of "No" on the no button
- **no_style** (*string*) – Optional style for the no button, defaults to "btn-primary"

Example:

```
bsmodals_confirm('Delete World',
    'Are you sure you want to delete the world', function(result) {
        if(result) {
            console.debug('User is despondent');
        }
        else {
            console.debug('Thankfully they said No');
        }
    });
```

Note that due to the optional paramters coming *after* the callback, this results in the unusual formatting of your code:

```
bsmodals_confirm('Chicken Type',
    'What kind of chicken do you want?', function(result) {
        if(result) {
            console.debug('They said Regular');
        }
        else {
            console.debug('They said Extra-Crispy');
        }
    }, yes_text='Regular', yes_style='btn-dark', no_text='Extra Crispy',
    no_style='btn-danger');
```

4.1.3 Custom Dialogs

Custom dialogs are instantiated through a class and inherit from `FormDialog`. To use a custom dialog you will need to include a template that extends a base template and then instantiate the class.

The `form` tag within your extending template is expected to use the `name` attribute on your `input`, `select`, and `textarea` tags. Methods are provided on the object for getting and setting the values of these tags. The `set_errors` method will put your tags into error mode, setting their `class` attribute to `is-invalid` and populating any nearby `<div class="invalid-feedback">` tags.

Base class methods:

`FormDialog.`**`set_data`**(*data*)
: Sets the contents of your your `input`, `select`, and `textarea` tags. Your tags must have the `name` attribute set corresponding to the keys in the data. The method correctly determines the use of `.val()` or `.text()` based on the tag type.

  Arguments

  - **`data`** (*`object`*) – Key/value pairs specifying the `name` and content of your tags.

`FormDialog.`**`get_data`**()
: Returns an object with key/value pairs corresponding to the contents of your form tags.

`FormDialog.`**`set_errors`**(*errors*)
: Changes your `input`, `select`, and `textarea` tags to be in the `is-invalid` state. If corresponding `<div class="invalid-feedback">` tags are nearby also sets their content to the error message.

  Arguments

  - **`errors`** (*`object`*) – Key/value pairs specifying the `name` and error message of any tags that are in error state.

FormModal

This class is for creating a dialog with a form inside. Use it by extending the `generic` template and filling the blocks with your `form`.

Example:

```
{% extends "bsmodals/generic.html" %}

{% block title %}
    <h5 id="mydialog-title">Dialog Title</h5>
{% endblock title %}

{% block body %}
    <form>
        <div class="form-group">
            <label for="name" class="col-form-label">Name</label>
            <input type="text" class="form-control" id="name">
        </div>
    </form>
{% endblock body %}

{% block footer %}
    <button id="mydialog-action" type="button" data-dismiss="modal"
        class="btn btn-primary">Close</button>
{% endblock footer %}
```

Inside of your HTML, include your newly written dialog using the `with` parameter of the `include` tag to set the dialog's id.

```
{% include "mydialog" with dialog_id="mydialog" %}
```

Once your template is in place, use the `FormModal` class to create an object, then call the `show` method to display the dialog.

```
var my_dialog = new FormModal('mydialog');
$('#mydialog-action').click(function() {
    console.debug('Somebody used MyDialog!');
});

var data = {
    'full_name':'Joe Smith'
}

my_dialog.show(data);
```

In the above example, the form element with the name attribute `full_name` is found and set to *Joe Smith* and the dialog is displayed. Note that the `generic` template does not include buttons in the dialog, so you will need to include these yourself and bind to them. The base template does include a block named `footer` that can be extended.

Class Definition:

FormModal(*dialog_id*)

Arguments

- **dialog_id** (*string*) – The id to use for your custom dialog, the helper function will search for this id to populate items in the dialog

FormModal.**show**(*data*)

Arguments

- **data** (*object*) – Key/value pairs in the object specify the contents of the `form` in the dialog box. `Form` tags such as `input`, `select` and `textarea` must have a `name` attribute corresponding to the key in the object.

AJAX Form

The `AJAXModal` class does everything the `FormModal` class does as well as submitting content values to a URL via `$.post`. A Django utility method is provided for managing the submission of the form in the Django view.

Sample view:

```
from django import forms
from django.http import JsonResponse
from bsmodals import handle_form


class SampleForm:
    name = forms.CharField(required=True)
    age = forms.IntegerField(required=True)


def ajax_form_view(request):
    form = SampleForm(request.POST)
    result, data = handle_form(form)

    if not result:
        print('Form contained errors! Returning them to the dialog')
        print('  => errors were:', data['errors'])

    return JsonResponse(data)
```

Corresponding form:

```
{% extends 'bsmodals/form.html' %}

{% block body %}
    <form>
        <div class="form-group">
            <label for="name" class="col-form-label">Name</label>
            <input type="text" class="form-control" name="name">
            <div class="invalid-feedback"></div>
        </div>
        <div class="form-group">
            <label for="age" class="col-form-label">Age</label>
            <input type="text" class="form-control" name="age">
            <div class="invalid-feedback"></div>
        </div>
    </form>
{% endblock body %}
```

And the javascript:

```
var my_form = new AJAXForm('myform');
var initial = {
```

(continues on next page)

(continued from previous page)

```
    'age':42,
}

my_form.show('/ajax_form_view', data, function(response) {
    if( response['success'] ) {
        console.log('Post succeeded. Dialog will now close');
    }
    else {
        // Post failed. The form fields now have "is-invalid" set and any
        // "invalid-feedback" <divs> now have the Django form errors
        // within them
        console.log('Post had errors');
    }
});
```

Class Definition

AJAXForm(*dialog_id*)

Constructs an object corresponding to an included Django template that extends `form`.

Arguments

- **dialog_id** (*string*) – The id to use for your custom dialog.

AJAXForm.**show**(*url*, *data*[, *callback=undefined*, *clear_on_success=true*])

Arguments

- **url** (*string*) – URL that the ajax POST is made to for form submission. Expects a JSON response, use the `handle_form()` helper method to generate it.
- **data** (*object*) – Key/value pairs to pre-populate the form with, uses the `set_data()` method to populate the form.
- **callback** (*function*) – Optional function to be called when the server responds to the post. Callback takes a parameter containing the JSON response.
- **clear_on_success** (*bool*) – Optional value that when false stops the values in the form being cleared after a successful submission. Defaults to true.

Server Side Helper

The Python `handle_form()` helper function can be used to validate the form and properly construct the JSON needed to be passed back to the form dialog.

bsmodals.**handle_form**(*form*)

Parameters **form** – Django form to be processed. Form field names should correspond to the `name` attributes of the fields in the HTML form.

Returns Tuple containing a boolean result and a dictionary to pass back via a JsonResponse object

Rest Forms

The `RestModal` class does everything the `FormModal` class does as well as providing methods for creating and updating objects using REST API URLs. To use the `RestModal` object you will need to include both the `bsmodals.`

`js` and `restapi.js` Javascript files in your HTML.

RestModal(*dialog_id*)

Parameters **dialog_id** (*string*) – The id to use for your custom dialog, the class will search for this id to populate items in the dialog

RestModal.**show_create**(*url*, *data*, *callback*)

Displays the dialog corresponding to the class and on submission calls the REST API POST method on the given URL to create the object.

Parameters

- **url** – URL of the REST POST call used to create objects represented by the form
- **data** – object whose key/value pairs are used to populate the form. Uses the inherited `set_data()` method.
- **callback** – optional method to call after the POST has succeeded

RestModal.**show_update**(*url*, *data*, *callback*)

Displays the dialog corresponding to the class and on submission calls the REST API PUT method on the given URL to update the object in question. Note that this URL must be for a single specific object.

Parameters

- **url** – URL for the REST API PUT call
- **data** – object whose key/value pairs are used to populate the form. Uses the inherited `set_data()` method.
- **callback** – optional method to call on success

RestModal.**show_patch**(*url*, *data*, *callback*)

Displays the dialog corresponding to the class and on submission calls the REST API PATCH method on the given URL to do a partial update on the object in question. Note that this URL must be for a single specific object.

Parameters

- **url** – URL for the REST API PUT call
- **data** – object whose key/value pairs are used to populate the form. Uses the inherited `set_data()` method.
- **callback** – optional method to call on success

4.1.4 Extra Parameters

Additional parameters can be set to change dialog behaviour. These parameters are set using the `with` parameter of the `include` tag.

not_centered By default all dialogs have the `modal-dialog-centered` Bootstrap class attribute which drops the dialog in the centre of the screen. Setting this value to `False` will remove the class attribute and the dialog will appear at the top.

no_click_off If true, turns off the closing of a dialog when clicking outside of it. Bootstrap calls this "static backdrop". Parameter is ignored with the `confirm` dialog.

modal_size (`generic` and `form` dialogs only) Alternate bootstrap dialog size specifier. Use things like `modal-lg` or `modal-xl` to add sizing info to the dialog.

title (`generic` and `form` dialogs only) Specify the title for the dialogs.

hide_cancel (`generic` dialogs only) Setting `hide_cancel` to `True` will prevent the `X` appearing in the top corner of the dialog box that closes the dialog.

4.1.5 Styling Dialogs

To provide additionaly styling to the base dialogs, their ids are as follows:

- Alert: `id="bsmodals-alert"`
- Error: `id="bsmodals-error"`
- Confirm: `id="bsmodals-confirm"`

The custom dialogs all define a body `block` for the contents of your form. The title is inside an `h5` tag with the id `{{dialog_id}}-title`.

The `generic` template defines a `footer` block that can be extended which is the ideal place for your dialog buttons.

Thee `form` template contains two buttons: `{{dialog_id}}-submit` and `{{dialog_id}}-cancel` for form submission and cancelling, respectively. See the section on *Extra Parameters* for other variables you can adjust.

4.1.6 Example Site

An example web-site is available with the source code:

Source: https://github.com/cltrudeau/django-bstrap-modals/tree/master/extras/sample_site

CHAPTER 5

Indices and tables

- genindex
- search

Index

A

AJAXForm() (*built-in function*), 14
AJAXForm.show() (*AJAXForm method*), 14

B

bsmodals.handle_form() (*built-in function*), 14
bsmodals_alert() (*built-in function*), 9
bsmodals_confirm() (*built-in function*), 10
bsmodals_error() (*built-in function*), 10

F

FormDialog.get_data() (*FormDialog method*), 11
FormDialog.set_data() (*FormDialog method*), 11
FormDialog.set_errors() (*FormDialog method*), 11
FormModal() (*built-in function*), 12
FormModal.show() (*FormModal method*), 13

R

RestModal() (*built-in function*), 15
RestModal.show_create() (*built-in function*), 15
RestModal.show_patch() (*built-in function*), 15
RestModal.show_update() (*built-in function*), 15