django-bootstrap-swt

Release v0.0.1

Jonas Kiefer

Jan 11, 2021

GETTING STARTED

Its features include:

- Permission checking before render the component.
- Provides auto id generating for bootstrap components.
- Includes the following bootstrap components: - Tooltip - Progressbar - Badge - Link - Button - Modal - Accordion - Card - Dropdown - more is coming soon...
- Javascript extensions to fetch content from url for Modal and Accordion component on opening/collapsing.

About the app:

- Available on pypi
- Documentation on readthedocs.org
- Bug tracker

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Installation

Django-bootstrap-swt is Available on pypi and can be installed using pip:

```
pip install django-bootstrap-swt
```

After installing, add `'django_bootstrap_swt'` to `INSTALLED_APPS`.

Also add all bootstrap 4 depending javascript and css content to your html template:

```
...
<head>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/
↪bootstrap.min.css" integrity="sha384-
↪Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin=
↪"anonymous">
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
↪KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin=
↪"anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.
↪min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/
↪ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"␣
↪integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"␣
↪crossorigin="anonymous"></script>    </head>
...
```

1.2 Examples

This shows how you can use the django-bootstrap-swt app by some examples.

For our examples we use the following models:

```
# tutorial/models.py
class Order(models.Model):
    name = models.CharField(max_length=100, verbose_name="full name")

class Product(models.Model):
    name = models.CharField(max_length=100, verbose_name="product name")
    order = models.ForeignKey(Order, related_name="items")
```

1.2.1 Use-Case: Provide action buttons for models.

So what if you like to provide the action buttons for all you can do with this models?

For this use case you can write a function under the model classes which returns the action buttons:

```
# tutorial/models.py
class Order(models.Model):
    ...
    def get_action_buttons(self):
        actions[LinkButton(url=self.edit_view_uri,
                           content='<i class="fas fa-edit"></i>',
                           color=ButtonColorEnum.WARNING,
                           tooltip=_l(f"Edit <strong>{self.name} [{self.id}]</strong>␣
↪Order."),
                           needs_perm=PermissionEnum.CAN_EDIT_ORDER.value)),
                LinkButton(url=self.delete_view_uri,
                           content='<i class="fas fa-trash-alt"></i>',
                           color=ButtonColorEnum.DANGER,
                           tooltip=_l(f"Delete <strong>{self.name} [{self.id}]</
↪strong> Order."),
                           needs_perm=PermissionEnum.CAN_DELETE_ORDER.value)),
                ]
```

On any view you then can call the *order.get_action_buttons()* function to get all possible actions.

1.2.2 Use-Case: Only renders buttons for that the user has permissions.

The next thing is, how can i only display the actions the user has permissions for?

For this use case you can use the *RenderHelper* class:

```
class OrderListView(ListView):
    model = Order
    ...
    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)

        permissions = self.request.user.get_all_permissions()
        user_permissions = []
        for permission in permissions:
            user_permissions.append(permission.codename)

        render_helper = RenderHelper(user_permissions=user_permissions)

        for order in context['object_list']:
            order.actions = render_helper.render_list_coherent(items=order.get_action_
↪buttons())

        return context
```

Now the rendered buttons are stored in your context and you can use them in your template like:

```
{% for object in object_list %}
    {{object.actions|safe}}
{% endfor %}
```

1.2.3 Use-Case: Update some attributes of the html component before rendering.

Sometimes you don't want to specify some attributes of a component at constructing time.

For this use case you can use again the *RenderHelper* class:

1. Update url query parameters. If you want to add or update some query parameters of your Action buttons you can do it with the following:

```
class OrderListView(ListView):
    model = Order
    ...
    def get_context_data(self, **kwargs):
        ...

        update_url_qs_dict = {'key-1': 'value-1', 'key-2': 'value-2'}

        render_helper = RenderHelper(user_permissions=user_permissions, update_url_
↪qs=update_url_qs_dict)

        for order in context['object_list']:
            order.actions = render_helper.render_list_coherent(items=order.get_action_
↪buttons())

        return context
```

All elements with an *href* attribute are updated like *http://example.com?key-1=value-1&key-2=value-2*.

2. Update the html attributes. You can also update any html attribute with the *RenderHelper*. For example you want to change the button size on every view:

```
class OrderListView(ListView):
    model = Order
    ...
    def get_context_data(self, **kwargs):
        ...

        update_attrs = {'class': ['btn-sm']}

        render_helper = RenderHelper(user_permissions=user_permissions, update_
↪attrs=update_attrs)

        for order in context['object_list']:
            order.actions = render_helper.render_list_coherent(items=order.get_action_
↪buttons())

        return context
```

1.3 Ajax components

Sometimes you need to fetch some content asynchronous. For this use case this app provides the possibility to setup an *fetch_url* on the *Modal* and *Accordion* component:

```
...
modal = Modal(title='nice modal',
              body='',
              btn_value='open modal',
              btn_color=ButtonColorEnum.INFO,
              fetch_url='http://example.com')
...
```

You also need to include our javascript bib to your html head:

```
<head>
    ...
    <script type="text/javascript" src="{% static 'django_bootstrap_swt/js/django_
↪bootstrap_swt.js' %}"></script>
    ...
</head>
```