Bambu AJAX Documentation

Release 2.0

Steadman

November 08, 2014

Contents

AJAX utility functions for Django that can be accessed via a single URL (ala WordPress admin-ajax.php)

CHAPTER 1

About Bambu AJAX

Bambu AJAX lets you write utility functions for your Django apps that can be called via AJAX, without having to specify a separate URL pattern and view for each one.

CHAPTER 2

About Bambu Tools 2.0

This is part of a toolset called Bambu Tools. It's being moved from a namespace of `bambu` to its own 'root-level' package, along with all the other tools in the set. If you're upgrading from a version prior to 2.0, please make sure to update your code to use `bambu_ajax` rather than `bambu.ajax`.

CHAPTER 3

Installation

Install the package via Pip:

```
pip install bambu-ajax
```

Add it to your `INSTALLED_APPS` list:

```
INSTALLED_APPS = (
    ...
    'bambu_ajax'
)
```

Add `bambu_ajax.urls` to your URLconf:

```
urlpatterns = patterns('',
    ...
    url(r'^ajax/', include('bambu_ajax.urls')),
)
```

Basic usage

Create a file called `ajax.php` within your Django app, and import the necessary module from the `bambu-ajax` package, like so:

```
from bambu_ajax import site

@site.register
def my_ajax_function(request):
    return [
        'a', 'list', 'of', 'things'
    ]
```

Using the `site.register` decorator registers your AJAX function with the `bambu_ajax` view.

To leverage this function from within a Django template, use the `ajaxurl` template tag, like so:

```
{% load ajax %}
<script>
    $.getJSON('{% ajaxurl 'my_project.my_app.my_ajax_function' %}&callback=?',
        function(data) {
            console.log(data);
        }
    );
</script>
```

Here, `my_project` should refer to the name of your Django project, `my_app` should be the name of the app you put your `ajax.py` file in, and `my_ajax_function` is the name of the function you defined within `ajax.py`.

The notation is similar to that used when referring to Django models, in that you always skip the common portion `ajax` from the naming convention.

CHAPTER 5

Shortcut

Make your life easier by including the utility library in your template:

```
<script src="{% url 'ajax_utility' %}"></script>
<script>
    bambu.ajax.get('my_project.my_app.my_ajax_function',
        function(data) {
            console.log(data);
        }
    );
</script>
```

This achieves the same result, but in a much cleaner way.

CHAPTER 6

Dynamically updating page content

As well as returning JSON-serialisable data per the examples above, you can also use AJAX functions just like views, in that they can return an `HttpResponse` object.

Using that method, the example above would print out the HTML (or other data) returned in the HTTP response.

CHAPTER 7

Additional decorators

You can of course add other decorators, just as you would with normal views.

CHAPTER 8

Todo

- Add a dedicated `login_required` decorator that returns a more helpful response for anonymous users
- Look into integrating this with Plunja, my dynamic JavaScript templating library.

CHAPTER 9

Questions or suggestions?

Find me on Twitter (@iamsteadman) or visit my blog.