django-siteajax Documentation

Release 1.0.0

Igor 'idle sign' Starikov

Jan 21, 2023

CONTENTS

https://github.com/idlesign/django-siteajax

CHAPTER

ONE

DESCRIPTION

Reusable application for Django bridging client and server sides

Streamline your server and client interaction using declarative techniques in your HTML and helpful abstractions from *siteajax* in your Python code.

Note: The client side of *siteajax* is powered by *htmx* (the successor of *intercooler.js*) - https://htmx.org/

CHAPTER

TWO

REQUIREMENTS

1. Python 3.7+
2. Django 2.2+

CHAPTER

THREE

TABLE OF CONTENTS

3.1 Quickstart

3.1.1 Installation

```
$ pip install django-siteajax
```

3.1.2 Configuration

In your project configuration file (usually `settings.py`):

1. Add `siteajax` to `INSTALLED_APPS`.
2. Add `siteajax.middleware.ajax_handler` to `MIDDLEWARE`.

> **Note:** The middleware is required if either your site is ajax-heavy (the majority of requests are ajax) or you do not use `@ajax_dispatch` decorator (it initializes `request.ajax` property just as middleware does) described below.

3.1.3 Basic Usage

Somewhere in your `views.py`:

```
from django.shortcuts import redirect, render
from siteajax.toolbox import Ajax, AjaxResponse


def index_page(request):
    """Suppose this view is served at /"""

    ajax: Ajax = request.ajax

    if ajax:

        news = ...  # Here we fetch some news from DB.

        response = render(request, 'mytemplates/sub_news.html', {'news': news})
```

(continues on next page)

(continued from previous page)

```
        # Now we can already return the response as usual.
        # But let's instruct the client side
        # to do some tricks. For that we use AjaxResponse wrapper:
        response = AjaxResponse(response)
        # Let's trigger `newsReady` event defined on client side
        # and pass some params into it:
        response.trigger_event(name='newsReady', kwargs={'count': len(news)})

        return response

    return render(request, 'mytemplates/index.html')
```

Now to your `mytemplates/index.html`:

```
<!DOCTYPE html>
<html>
<head>
    <!-- Get client library js from CDN. -->
    {% include "siteajax/cdn.html" %}
</head>
<body>
    <div id="news-list" hx-get hx-trigger="load"></div>
    <!-- The contents of the above div will be replaced
        with news from server automatically fetched on page load. -->

    <!-- Initialize CSRF token for Django (if you ever want to use POST/PUT etc.) -->
    <script>{% include "siteajax/init_csrf.js" %}</script>

    <script>
        document.body.addEventListener('newsReady', function(event){
           alert('News loaded: ' + event.detail.count);
        })
    </script>

</body>
</html>
```

At last `mytemplates/sub_news.html` (nothing special):

```
{% for item in news %}<div>{{ item.title }}</div>{% endfor %}
```

Note: See https://htmx.org/docs/ for more examples of client side

Note: See https://github.com/idlesign/django-siteajax/tree/master/demo for *siteajax* usage example.

3.1.4 Dispatch

In cases when various ajax calls have a single entry point view it's useful to apply `siteajax.toolbox.Ajax.ajax_dispatch` decorator to decouple logic.

It allows ajax request dispatch based on source html element identifiers. So the above mentioned `index_page` entry point view can be defined as follows

```
from django.shortcuts import redirect, render
from siteajax.toolbox import AjaxResponse, ajax_dispatch


def get_news(request):
    news = ...  # Here we fetch some news from DB.
    response = AjaxResponse(render(request, 'mytemplates/sub_news.html', {'news': news}))
    response.trigger_event(name='newsReady', kwargs={'count': len(news)})
    return response

@ajax_dispatch({
    'news-list': get_news,  # Map element id to a handler
})
def index_page(request):
    """Suppose this view is served at /"""
    return render(request, 'mytemplates/index.html')
```

For cases when you want to route multiple autogenerated page elements with IDs having a common prefix (e.g.: in `myel-1`, `myel-44`, `myel-something` common prefix is `myel-`) to a single handler use star (*):

```
@ajax_dispatch({
    'myel-*': common_handler,
})
def my_iew(request):
    ...
```

3.2 Advanced

3.2.1 Additional info from the client

`request.ajax` object is `siteajax.toolbox.Ajax`. It given you an access to additional information received from the client:

- `ajax.is_used` - a flag indicating whether the request is Ajax or not
- `ajax.restore_history` - indicates the client side is requesting the entire page (as opposed to a page fragment request), when the client was unable to restore a browser history state from the cache.
- `ajax.url` - URL from the browser
- `ajax.target` - the id of the target element if it exists
- `ajax.user_input` - user input given to a prompt (hx-prompt)
- `ajax.source` - info about an element sourcing (triggered) the request (`id` and `name` if any)

Note: The object is lazily initialized to allow faster middleware processing.

Without initialization you won't be able to access it's attributes.

For the initialization it's enough to check it in boolean context, e.g.:

```
bool(Ajax(request))

# or

if request.ajax:
    ...
```

3.2.2 Driving the client

Wrap your response into `siteajax.toolbox.AjaxResponse` to be able to instruct your client to do thing:

```
from django.shortcuts import render
from siteajax.toolbox import Ajax, AjaxResponse


def index_page(request):

    response = render(request, 'some.html')

    # Wrap it
    response = AjaxResponse(response)

    # Let's trigger `fireThis` event after `swap` step
    response.trigger_event(name='fireThis', kwargs={'count': len(news)}, step='swap')

    # Add an item to browser history
    response.history_item = '/otherurl/'

    # Redirect with JS
    response.redirect = '/here/'

    # Refresh current page
    response.refresh = True

    return response
```

3.2.3 CSRF protection

Include `siteajax/init_csrf.js` in your template (page's `body`) to initialize CSRF token required to `POST`, `PUT`, `DELETE`.

```
<script>{% include "siteajax/init_csrf.js" %}</script>
```

3.2.4 Include htmx from CDN

You can make use of including `siteajax/cdn.html` in your template (page's `head`) to get `htmx` right from a CDN.

```
{% include "siteajax/cdn.html" %}
```

Note: If you're not satisfied with the version included you can always define your own `<script src=`.