django-htmx Documentation

Release 1.17.3

Adam Johnson

Apr 22, 2024

CONTENTS:

Extensions for using Django with htmx.

CHAPTER

ONE

INSTALLATION

1.1 Requirements

Python 3.8 to 3.12 supported.

Django 3.2 to 5.0 supported.

1.2 Installation

1. Install with **pip**:

```
python -m pip install django-htmx
```

2. Add django-htmx to your `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    ...,
    "django_htmx",
    ...,
]
```

3. Add the middleware:

```
MIDDLEWARE = [
    ...,
    "django_htmx.middleware.HtmxMiddleware",
    ...,
]
```

   The middleware adds `request.htmx`, as described in *Middleware*.

4. (Optional) Add the extension script to your base template, as documented in *Extension Script*.

1.3 Install htmx

django-htmx does not include htmx itself, since it can work with many different versions. It's up to you to add htmx (and any extensions) to your project.

> **Warning: JavaScript CDN's**
>
> Avoid using JavaScript CDN's like unpkg.com to include htmx (or any resources). They reduce privacy, performance, and security - see this post.

You can add htmx like so:

1. Download `htmx.min.js` from its latest release.
2. Put `htmx.min.js` in a static directory in your project. For example, if you have a `static/` directory in your `STATICFILES_DIRS` setting:

   ```
   STATICFILES_DIRS = [BASE_DIR / "static"]
   ```

   …then put it in there, organized as you like, such as in a `js/` sub-directory.
3. Add a <script> tag in your base template, within your <head>:

   ```
   {% load static %}
   <script src="{% static 'htmx.min.js' %}" defer></script>
   ```

   (or `js/htmx.min.js`, etc.).

   The `defer` attribute allows the browser to continue rendering the page whilst htmx is downloading, making your site's first render faster.

   If you have multiple base templates for pages that you want htmx on, add the <script> on all of them.

Note: Extensions

You can adapt the above steps to set up htmx's extensions that you wish to use. Download them from htmx's `ext/` folder into your project, and include their script tags after htmx, for example:

```
{% load static %}
<script src="{% static 'js/htmx/htmx.min.js' %}" defer></script>
<script src="{% static 'js/htmx/debug.js' %}" defer></script>
```

CHAPTER

TWO

MIDDLEWARE

class django_htmx.middleware.**HtmxMiddleware**

This middleware attaches `request.htmx`, an instance of *HtmxDetails* (below). Your views, and any following middleware, can use `request.htmx` to switch behaviour for requests from htmx. The middleware supports both sync and async modes.

See it action in the "Middleware Tester" section of the *example project*.

class django_htmx.middleware.**HtmxDetails**

This class provides shortcuts for reading the htmx-specific request headers.

__bool__()

`True` if the request was made with htmx, otherwise `False`. Detected by checking if the `HX-Request` header equals `true`.

This behaviour allows you to switch behaviour for requests made with htmx:

```
def my_view(request):
    if request.htmx:
        template_name = "partial.html"
    else:
        template_name = "complete.html"
    return render(template_name, ...)
```

Return type
bool

boosted: bool

`True` if the request came from an element with the `hx-boost` attribute. Detected by checking if the `HX-Boosted` header equals `true`.

You can use this attribute to change behaviour for boosted requests:

```
def my_view(request):
    if request.htmx.boosted:
        # do something special
        ...
    return render(...)
```

current_url: str | None

The current URL in the browser that htmx made this request from, or `None` for non-htmx requests. Based on the `HX-Current-URL` header.

current_url_abs_path: str | None

The absolute-path form of `current_url`, that is the URL without scheme or netloc, or `None` for non-htmx requests.

This value will also be `None` if the scheme and netloc do not match the request. This could happen if the request is cross-origin, or if Django is not configured correctly.

For example:

```
>>> request.htmx.current_url
'https://example.com/dashboard/?year=2022'
>>> # assuming request.scheme and request.get_host() match:
>>> request.htmx.current_url_abs_path
'/dashboard/?year=2022'
```

This is useful for redirects:

```
if not sudo_mode_active(request):
    next_url = request.htmx.current_url_abs_path or ""
    return HttpResponseClientRedirect(f"/activate-sudo/?next={next_url}")
```

history_restore_request: bool

`True` if the request is for history restoration after a miss in the local history cache. Detected by checking if the `HX-History-Restore-Request` header equals `true`.

prompt: str | None

The user response to hx-prompt if it was used, or `None`.

target: str | None

The `id` of the target element if it exists, or `None`. Based on the `HX-Target` header.

trigger: str | None

The `id` of the triggered element if it exists, or `None`. Based on the `HX-Trigger` header.

trigger_name: str | None

The `name` of the triggered element if it exists, or `None`. Based on the `HX-Trigger-Name` header.

triggering_event: Any | None

The deserialized JSON representation of the event that triggered the request if it exists, or `None`. This header is set by the event-header htmx extension, and contains details of the DOM event that triggered the request.

CHAPTER

THREE

EXTENSION SCRIPT

django-htmx comes with a small JavaScript extension for htmx's behaviour. Currently the extension only includes a debug error handler, documented below.

3.1 Installation

The script is served as a static file called `django-htmx.js`, but you shouldn't reference it directly. Instead, use the included template tags, for both Django and Jinja templates.

3.1.1 Django Templates

Load and use the template tag in your base template, after your htmx `<script>` tag:

```
{% load django_htmx %}
<!doctype html>
<html>
  ...
  <script src="{% static 'js/htmx.min.js' %}" defer></script>{# or however you include␣
↪htmx #}
  {% django_htmx_script %}
  </body>
</html>
```

3.1.2 Jinja Templates

First, load the tag function into the globals of your custom environment:

```
# myproject/jinja2.py
from jinja2 import Environment
from django_htmx.jinja import django_htmx_script


def environment(**options):
    env = Environment(**options)
    env.globals.update(
        {
            # ...
            "django_htmx_script": django_htmx_script,
```

(continues on next page)

(continued from previous page)

```
        }
    )
    return env
```

Second, call the function in your base template, after your htmx <script> tag:

```
{{ django_htmx_script() }}
```

3.2 Debug Error Handler

htmx's default behaviour when encountering an HTTP error is to discard the response content. This can make it hard to debug errors in development.

The django-htmx script includes an error handler that's active when Django's debug mode is on (`settings.DEBUG` is `True`). The handler detects responses with 404 and 500 status codes and replaces the page with their content. This change allows you to debug with Django's default error responses as you would for a non-htmx request.

See this in action in the "Error Demo" section of the *example project*.

Hint: This extension script should not be confused with htmx's debug extension, which logs DOM events in the browser console.

CHAPTER

FOUR

HTTP TOOLS

4.1 Response classes

class django_htmx.http.**HttpResponseClientRedirect**(*redirect_to*, **args*, ***kwargs*)

htmx can trigger a client side redirect when it receives a response with the HX-Redirect header. HttpResponseClientRedirect is a HttpResponseRedirect subclass for triggering such redirects.

Parameters

- **redirect_to** (*str*) – The path to redirect to, as per HttpResponseRedirect.
- **args** (*Any*) – Other HTTPResponse parameters.
- **kwargs** (*Any*) – Other HTTPResponse parameters.

For example:

```
from django_htmx.http import HttpResponseClientRedirect


def sensitive_view(request):
    if not sudo_mode.active(request):
        return HttpResponseClientRedirect("/activate-sudo-mode/")
    ...
```

class django_htmx.http.**HttpResponseClientRefresh**

htmx will trigger a page reload when it receives a response with the HX-Refresh header. HttpResponseClientRefresh is a custom response class that allows you to send such a response. It takes no arguments, since htmx ignores any content.

For example:

```
from django_htmx.http import HttpResponseClientRefresh


def partial_table_view(request):
    if page_outdated(request):
        return HttpResponseClientRefresh()
    ...
```

class django_htmx.http.**HttpResponseLocation**(*redirect_to*, **args*, *source=None*, *event=None*, *target=None*, *swap=None*, *values=None*, *headers=None*, ***kwargs*)

An HTTP response class for sending the HX-Location header. This header makes htmx make a client-side "boosted" request, acting like a client side redirect with a page reload.

Parameters

- **redirect_to** (*str*) – The path to redirect to, as per HttpResponseRedirect.
- **source** (*str | None*) – The source element of the request.
- **event** (*str | None*) – The event that "triggered" the request.
- **target** (*str | None*) – CSS selector to target.
- **swap** (*Literal['innerHTML', 'outerHTML', 'beforebegin', 'afterbegin', 'beforeend', 'afterend', 'delete', 'none', None]*) – How the response will be swapped into the target.
- **values** (*dict[str, str] | None*) – values to submit with the request.
- **headers** (*dict[str, str] | None*) – headers to submit with the request.
- **args** (*Any*) – Other HTTPResponse parameters.
- **kwargs** (*Any*) – Other HTTPResponse parameters.

For example:

```
from django_htmx.http import HttpResponseLocation


def wait_for_completion(request, action_id):
    ...
    if action.completed:
        return HttpResponseLocation(f"/action/{action.id}/completed/")
    ...
```

class django_htmx.http.**HttpResponseStopPolling**(**args*, ***kwargs*)

When using a polling trigger, htmx will stop polling when it encounters a response with the special HTTP status code 286. HttpResponseStopPolling is a custom response class with that status code.

Parameters

- **args** (*Any*) – Other HTTPResponse parameters.
- **kwargs** (*Any*) – Other HTTPResponse parameters.

For example:

```
from django_htmx.http import HttpResponseStopPolling


def my_pollable_view(request):
    if event_finished():
        return HttpResponseStopPolling()
    ...
```

django_htmx.http.**HTMX_STOP_POLLING: int = 286**

A constant for the HTTP status code 286. You can use this instead of `HttpResponseStopPolling` to stop htmx from polling.

For example, with Django's render shortcut:

```
from django_htmx.http import HTMX_STOP_POLLING


def my_pollable_view(request):
    if event_finished():
        return render("event-finished.html", status=HTMX_STOP_POLLING)
    ...
```

4.2 Response modifying functions

django_htmx.http.**push_url**(*response*, *url*)

Set the HX-Push-Url header of `response` and return it. This header makes htmx push the given URL into the browser location history.

Parameters

- **response** (*_HttpResponse*) – The response to modify and return.
- **url** (*str | Literal[False]*) – The (relative) URL to push, or `False` to prevent the location history from being updated.

Return type

_HttpResponse

For example:

```
from django_htmx.http import push_url


def leaf(request, leaf_id):
    ...
    if leaf is None:
        # Directly render branch view
        response = branch(request, branch=leaf.branch)
        return push_url(response, f"/branch/{leaf.branch.id}")
    ...
```

django_htmx.http.**reswap**(*response*, *method*)

Set the HX-Reswap header of `response` and return it. This header overrides the swap method that htmx will use.

Parameters

- **response** (*_HttpResponse*) – The response to modify and return.
- **method** (*str*) – The swap method.

Return type

_HttpResponse

For example:

```
from django_htmx.http import reswap


def employee_table_row(request):
    ...
    response = render(...)
    if employee.is_boss:
        reswap(response, "afterbegin")
    return response
```

django_htmx.http.**retarget**(*response*, *target*)

Set the HX-Retarget header of `response` and return it. This header overrides the element that htmx will swap content into.

Parameters

- **response** (*_HttpResponse*) – The response to modify and return.
- **target** (*str*) – CSS selector to target.

Return type
_HttpResponse

For example:

```
from django.views.decorators.http import require_POST
from django_htmx.http import retarget


@require_POST
def add_widget(request):
    ...

    if form.is_valid():
        # Rerender the whole table on success
        response = render("widget-table.html", ...)
        return retarget(response, "#widgets")

    # Render just inline table row on failure
    return render("widget-table-row.html", ...)
```

django_htmx.http.**trigger_client_event**(*response*, *name*, *params=None*, ***, *after='receive'*, *encoder=<class 'django.core.serializers.json.DjangoJSONEncoder'>*)

Modify one of the HX-Trigger headers of `response` and return it. These headers make htmx trigger client-side events.

Calling `trigger_client_event` multiple times for the same `response` and `after` will update the appropriate header, preserving existing event specifications.

Parameters

- **response** (*_HttpResponse*) – The response to modify and return.
- **name** (*str*) – The name of the event to trigger.
- **params** (*dict[str, Any] | None*) – Optional JSON-compatible parameters for the event.

- **after** (*Literal['receive', 'settle', 'swap']*) – Which `HX-Trigger` header to modify:
  - `"receive"`, the default, maps to `HX-Trigger`
  - `"settle"` maps to `HX-Trigger-After-Settle`
  - `"swap"` maps to `HX-Trigger-After-Swap`
- **encoder** (*type[JSONEncoder]*) – The `JSONEncoder` class used to generate the JSON. Defaults to `DjangoJSONEncoder` for its extended data type support.

Return type
_HttpResponse

For example:

```
from django_htmx.http import trigger_client_event


def end_of_long_process(request):
    response = render("end-of-long-process.html")
    return trigger_client_event(
        response,
        "showConfetti",
        {"colours": ["purple", "red", "pink"]},
        after="swap",
    )
```

CHAPTER

FIVE

EXAMPLE PROJECT

The django-htmx repository contains an example project, demonstrating use of django-htmx. Run it locally and check out its source code to learn about using htmx with Django, and how django-htmx can help.

CHAPTER

SIX

TIPS

This page contains some tips for using htmx with Django.

6.1 Make htmx Pass the CSRF Token

If you use htmx to make requests with "unsafe" methods, such as POST via hx-post, you will need to make htmx cooperate with Django's Cross Site Request Forgery (CSRF) protection. Django can accept the CSRF token in a header, normally *X-CSRFToken* (configurable with the `CSRF_HEADER_NAME` setting, but there's rarely a reason to change it).

You can make htmx pass the header with its `hx-headers` attribute. It's most convenient to place `hx-headers` on your `<body>` tag, as then all elements will inherit it. For example:

```
<body hx-headers='{"X-CSRFToken": "{{ csrf_token }}"}'>
   ...
</body>
```

Note this uses `{{ csrf_token }}`, the variable, as opposed to `{% csrf_token %}`, the tag that renders a hidden `<input>`.

This snippet should work with both Django templates and Jinja.

For an example of this in action, see the "CSRF Demo" page of the *example project*.

6.2 Partial Rendering

For requests made with htmx, you may want to reduce the page content you render, since only part of the page gets updated. This is a small optimization compared to correctly setting up compression, caching, etc.

You can use Django's template inheritance to limit rendered content to only the affected section. In your view, set up a context variable for your base template like so:

```
from django.http import HttpRequest, HttpResponse
from django.shortcuts import render
from django.views.decorators.http import require_GET


@require_GET
def partial_rendering(request: HttpRequest) -> HttpResponse:
    if request.htmx:
        base_template = "_partial.html"
```

(continues on next page)

(continued from previous page)

```
    else:
        base_template = "_base.html"

    ...

    return render(
        request,
        "page.html",
        {
            "base_template": base_template,
            # ...
        },
    )
```

Then in the template (`page.html`), use that variable in `{% extends %}`:

```
{% extends base_template %}

{% block main %}
  ...
{% endblock %}
```

Here, `_base.html` would be the main site base:

```
<!doctype html>
<html>
<head>
  ...
</head>
<body>
  <header>
    <nav>
      ...
    </nav>
  </header>
  <main id="main">
    {% block main %}{% endblock %}
  </main>
</body>
```

…whilst `_partial.html` would contain only the minimum element to update:

```
<main id="main">
  {% block main %}{% endblock %}
</main>
```

For an example of this in action, see the "Partial Rendering" page of the *example project*.

CHAPTER

SEVEN

CHANGELOG

7.1 1.17.3 (2024-03-01)

- Change `reswap()` type hint for `method` to `str`.

  Thanks to Dan Jacob for the report in Issue #421 and fix in PR #422.

7.2 1.17.2 (2023-11-16)

- Fix asgiref dependency declaration.

7.3 1.17.1 (2023-11-14)

- Fix ASGI compatibility on Python 3.12.

  Thanks to Grigory Vydrin for the report in Issue #381.

7.4 1.17.0 (2023-10-11)

- Support Django 5.0.

7.5 1.16.0 (2023-07-10)

- Drop Python 3.7 support.
- Remove the unnecessary `type` attribute on the `<script>` tag generated by `django_htmx_script`.
- Allow custom JSON encoders in `trigger_client_event()`.

  Thanks to Joey Lange in PR #349.

7.6 1.15.0 (2023-06-13)

- Support Python 3.12.

7.7 1.14.0 (2023-02-25)

- Support Django 4.2.

7.8 1.13.0 (2022-11-10)

- Make the `params` argument of *`trigger_client_event()`* optional.

  Thanks to Chris Tapper in PR #263.
- Add *`django_htmx.http.push_url()`* for setting the `HX-Push-URL` header.

  Thanks to Chris Tapper in PR #264.
- Add *`django_htmx.http.reswap()`* for setting the `HX-Reswap` header added in htmx 1.8.0.
- Add *`django_htmx.http.retarget()`* for setting the `HX-Retarget` header added in htmx 1.6.1.
- Add *`HttpResponseLocation`* for sending a response with the `HX-Location` header.

  Thanks to Ben Beecher in PR #239.
- Add *`request.htmx.current_url_abs_path`*, the absolute-path form of `request.current_url`.

  Thanks to Trey Hunner for the feature request in Issue #259.

7.9 1.12.2 (2022-08-31)

- Improve type hints for `trigger_client_event()` by using a `TypeVar`.

  Thanks to Chris Tapper in PR #260.

7.10 1.12.1 (2022-07-29)

- Override `HttpResponseClientRedirect.url` property to fix `HttpResponseClientRedirect.__repr__`.

7.11 1.12.0 (2022-06-05)

- Support Python 3.11.
- Support Django 4.1.

7.12 1.11.0 (2022-05-10)

- Drop support for Django 2.2, 3.0, and 3.1.

7.13 1.10.0 (2022-05-07)

- Make `trigger_client_event()` return the response.
- Add async support to `HtmxMiddleware` to reduce overhead on async views.

7.14 1.9.0 (2022-03-02)

- Move documentation from the README to Read the Docs. Also expand it with sections on installing htmx, and configuring CSRF.

  Thanks to Ben Beecher for intial setup in PR #194.
- Add `HttpResponseClientRefresh` for telling htmx to reload the page.

  Thanks to Bogumil Schube in PR #193.

7.15 1.8.0 (2022-01-10)

- Drop Python 3.6 support.

7.16 1.7.0 (2022-01-10)

- Use `DjangoJSONEncoder` for encoding the `HX-Trigger` event.

  Thanks to Cleiton de Lima in PR #182.
- Drop redundant 'async' from debug `<script>` tag.

7.17 1.6.0 (2021-10-06)

- Add `HttpResponseClientRedirect` class for sending HTMX client-side redirects.

  Thanks to Julio César in PR #121.
- Add `django_htmx.http.trigger_client_event()` for triggering client side events.

7.18 1.5.0 (2021-10-05)

- Support Python 3.10.

7.19 1.4.0 (2021-10-02)

- Support the `HX-Boosted` header, which was added in htmx 1.6.0. This is parsed into the `request.htmx.boosted` attribute.

7.20 1.3.0 (2021-09-28)

- Support Django 4.0.

7.21 1.2.1 (2021-07-09)

- Make extension script error handler also show 404 errors.

7.22 1.2.0 (2021-07-08)

- Installation now requires adding `"django_htmx"` to your `INSTALLED_APPS` setting.
- Add extension script with debug error handler. To install it, follow the new instructions in the README.

  htmx's default behaviour is to discard error responses. The extension overrides this in debug mode to shows Django's debug error responses.
- Add `django_htmx.http` module with `HttpResponseStopPolling` class and `HTMX_STOP_POLLING` constant.

7.23 1.1.0 (2021-06-03)

- Support the `HX-History-Restore-Request` header, which was added in htmx 1.2.0. This is parsed into the `request.htmx.history_restore_request` attribute.
- Support the `Triggering-Event` header, which is sent by the event-header extension. This is parsed into the `request.htmx.triggering_event` attribute.
- Stop distributing tests to reduce package size. Tests are not intended to be run outside of the tox setup in the repository. Repackagers can use GitHub's tarballs per tag.

7.24 1.0.1 (2021-02-08)

- Remove `X-HTTP-Method-Override` handling from `HtmxMiddleware`. This has not been needed since htmx 0.0.5, when use of the header was extracted to its `method-override` extension in htmx commit 2305ae.

7.25 1.0.0 (2021-02-07)

- Add `HtmxMiddleware` which handles request headers from htmx.
- Add example app on GitHub repository which demonstrates using django-htmx features.
- Remove the `{% htmx_script %}` template tag. Include htmx on your pages yourself - this allows you to better customize the way htmx is installed to suit your project - for example by using the `async` script attribute or by bundling it with extensions.
- Remove the `HTMXViewMixin`, `{% htmx_include %}` and `{% htmx_attrs %}` tags. Partial rendering can be done more with a simpler techinque - see the demo page in the example app, added in Pull Request #30.

7.26 0.1.4 (2020-06-30)

- This version and those before explored what's possible with htmx and django, but were not documented.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

B

C

H

P

R

T