Make WebDev fun again through HTMX with Django

Rexsy Bima Trima Wahyu

github.com/RexsyBima

Repo : github.com/RexsyBima/pyconid2025-django-htmx

Who Am I

Freelancer (Django for 3 years or so)

Teach Python Privately Outside of Indonesia (english speakers main)

Neovim enjoyer

ONCE a open source contributor for Omarchy

The problem or challenge In webdev

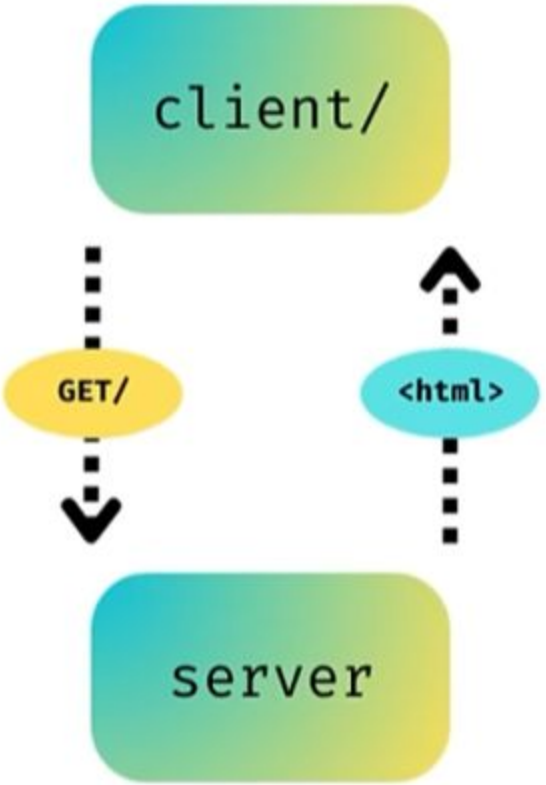
1. Classic: just too many ecosystem to build a webapp
2. We (as a python dev) want something that can get the job done
3. Personal take: Im more of a backend guy :)
4. 2nd personal take: I know Python more than JS

HTMX in a Nutshell

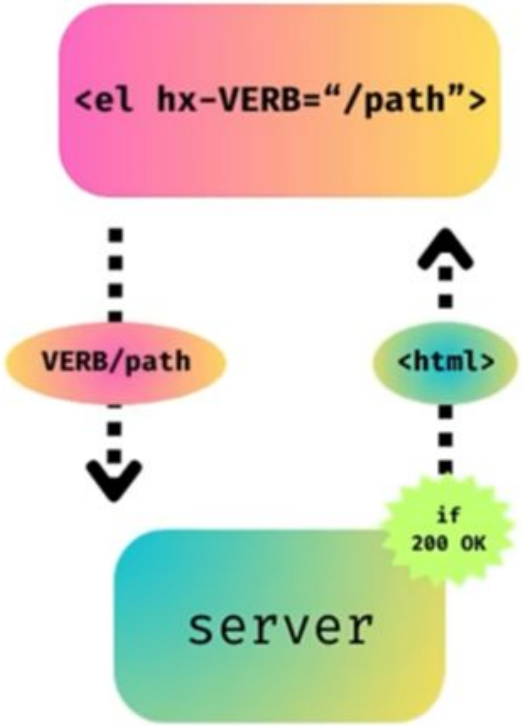
htmx is a library that allows you to access modern browser features directly from HTML, rather than using javascript.

HTMX can be applied to every server framework that can handle a HTTP Response

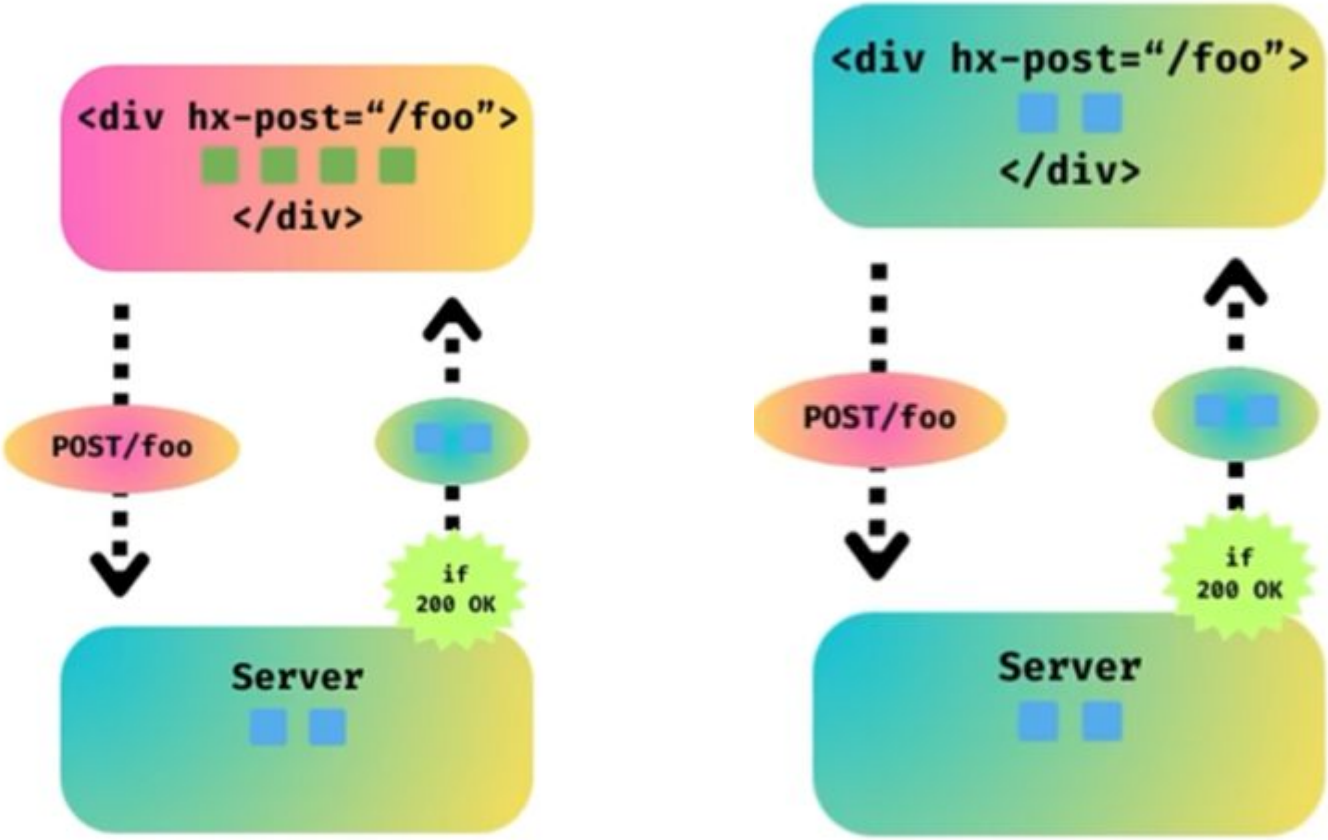
STANDARD SERVER INTERACTION



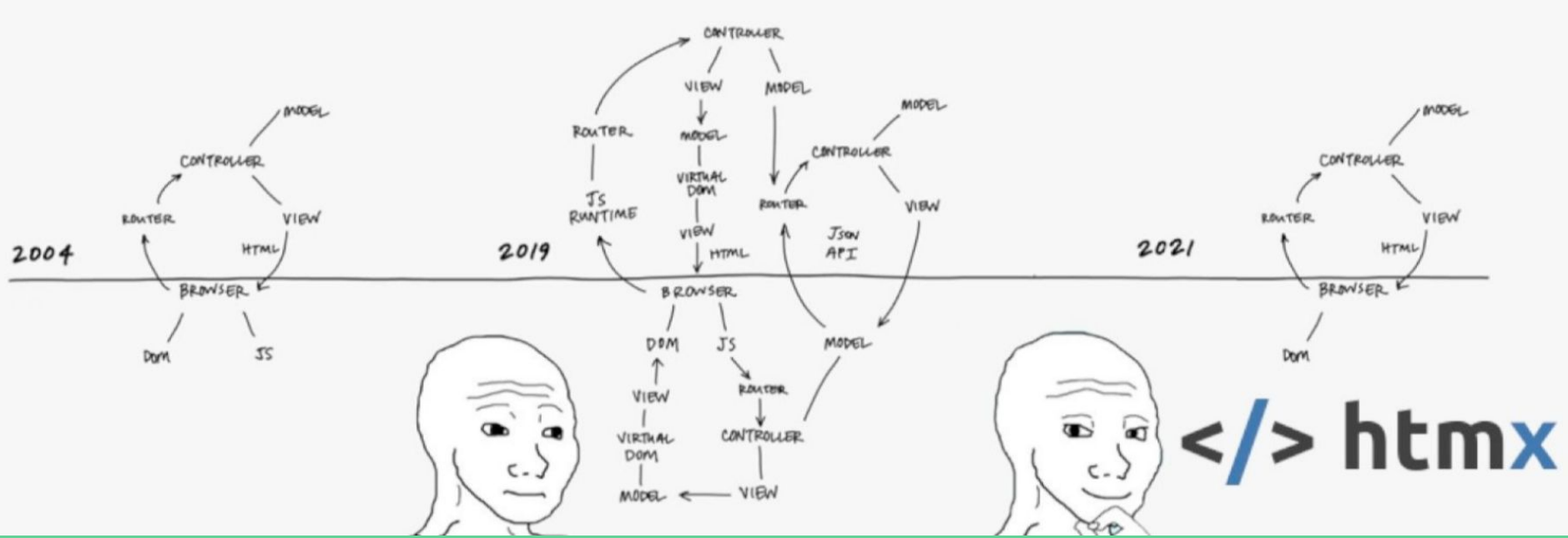
</> htmx



</> htmx



Classic Meme



HTMX Motivation

motivation

- Why should only `<a>` & `<form>` be able to make HTTP requests?
- Why should only `click` & `submit` events trigger them?
- Why should only `GET` & `POST` methods be available?
- Why should you only be able to replace the **entire** screen?

Installation

Installation

Installing

Htmx is a dependency-free, browser-oriented javascript library. This means that using it is as simple as adding a `<script>` tag to your document head. There is no need for a build system to use it.

1. CDN
2. Download the htmx.js into our system

Check more at https://htmx.org/docs/#installing

Download a copy

The next easiest way to install htmx is to simply copy it into your project.

Download `htmx.min.js` from jsDelivr and add it to the appropriate directory in your project and include it where necessary with a `<script>` tag:

```
<script src="/path/to/htmx.min.js"></script>
```

core/base.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">

<head>
    <script src={% static "js/htmx.min.js" %}></script>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Django HTMX</title>
</head>

<body>
    {% block body %}
    {% endblock body %}
</body>

</html>
```

HTMX in a Nutshell

views.py

```python
from django.http import HttpRequest, HttpResponse
from django.shortcuts import render


# Create your views here.
def hello_pycon(request: HttpRequest):
    return render(
        request,
        "core/hello_pycon.html",
    )


def get_hello_pycon_answer(request: HttpRequest):
    return HttpResponse("Hello Pycon ID 2025")
```

HTMX in a Nutshell

core/hello_pycon.html

```
{% extends "base.html" %}
{% block body %}
<button hx-get="{% url 'get_hello_pycon_answer' %}" hx-trigger="click" hx-target="#target-output"
  hx-swap="innerHTML">Click Me!</button>

<h1 id="target-output"></h1>
{% endblock body %}
```

On the button, it reads as

“When a user clicks on this button, issue an HTTP GET request to url ‘get_hello_pycon_answer’ and use the content from the response to replace the inner element with the id “target-output” in the DOM”

SMALL DEMO ->
http://localhost:8000/hello_pycon/

CORE CONCEPTS OF HTMX

hx-get*, hx-trigger, hx-target, hx-swap

(AJAX) hx-get*

Attribute	Description
hx-get	Issues a `GET` request to the given URL
hx-post	Issues a `POST` request to the given URL
hx-put	Issues a `PUT` request to the given URL
hx-patch	Issues a `PATCH` request to the given URL
hx-delete	Issues a `DELETE` request to the given URL

(AJAX) hx-get*

```
<button hx-put="/messages">
    Put To Messages
</button>
```

This tells the browser:

> When a user clicks on this button, issue a PUT request to the URL /messages and load the response into the button

hx-trigger

Triggering Requests

By default, AJAX requests are triggered by the “natural” event of an element:

- `input`, `textarea` & `select` are triggered on the `change` event
- `form` is triggered on the `submit` event
- everything else is triggered by the `click` event

If you want different behavior you can use the hx-trigger attribute to specify which event will cause the request.

hx-target

Targets

If you want the response to be loaded into a different element other than the one that made the request, you can use the hx-target attribute, which takes a CSS selector. Looking back at our Live Search example:

```
<input type="text" name="q"
    hx-get="/trigger_delay"
    hx-trigger="keyup delay:500ms changed"
    hx-target="#search-results"
    placeholder="Search...">
<div id="search-results"></div>
```

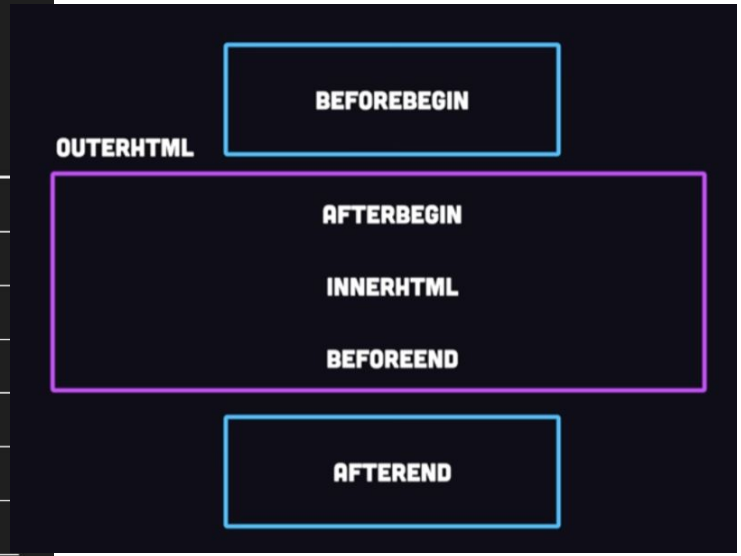
You can see that the results from the search are going to be loaded into `div#search-results`, rather than into the input tag.

hx-swap

Swapping

htmx offers a few different ways to swap the HTML returned into the DOM. By default, the content replaces the `innerHTML` of the target element. You can modify this by using the hx-swap attribute with any of the following values:

Name	Description
`innerHTML`	the default, puts the content inside the target element
`outerHTML`	replaces the entire target element with the returned content
`afterbegin`	prepends the content before the first child inside the target
`beforebegin`	prepends the content before the target in the target's parent element
`beforeend`	appends the content after the last child inside the target
`afterend`	appends the content after the target in the target's parent element
`delete`	deletes the target element regardless of the response
`none`	does not append content from response (Out of Band Swaps and Response Headers will still be processed)



MORE DEMO

http://localhost:8000/todos/all/

http://localhost:8000/todos/all_htmx/

Starting Page

todo/views.py

```
def get_all_todos_htmx(request: HttpRequest):
    todos = Todo.objects.all()
    return render(
        request, "todo/all_todos_htmx.html", {"todos": todos, "count": len(todos)}
    )
```

The Templates

todo/all_todos_htmx.html

```
{% extends "base.html" %}
{% block body %}
<div id="todo-list">
  {% for todo in todos %}
  <h1 id="todo-{{todo.id}}">
    {{ todo.name }}
    <a hx-get="{% url 'get_todo_htmx' todo.id %}" hx-target="#todo-tooltips">get detail</a>
    <a hx-get="{% url 'update_todo_htmx' todo.id %}" hx-target="#todo-tooltips">update todo</a>
    <a hx-delete="{% url 'delete_todo_htmx' todo.id %}" hx-target="#todo-{{todo.id}}" hx-swap="outerHTML"
      hx-confirm="are you sure you want to delete this todo {{ todo.name }}?">delete todo</a>
  </h1>

  {% endfor %}
</div>

<a hx-get="{% url 'create_todo_htmx' %}" hx-target="#todo-tooltips">create todo v1</a>
<a hx-get="{% url 'create_todo_htmx_v2' %}" hx-target="#todo-tooltips">create todo v2</a>
<h1 id="todos-length">Todos length: {{ count }}</h1>

<div id="todo-tooltips"></div>


{% endblock body %}
```

get_todo

todo/views.py

“from django.shortcuts import get_object_or_404”

```python
def get_todo_htmx(request: HttpRequest, pk: int):
    todo = get_object_or_404(Todo, pk=pk)
    return render(request, "todo/todo_htmx.html", {"todo": todo})
```

Template response

todo/todo_htmx.html

```
<h2>{{ todo.name }}</h2>
<h1>{{ todo.description }}</h1>

<h1 hx-delete="{% url 'delete_element' %}" hx-target="#todo-tooltips">Close Detail!</h1>
```

from django.views.decorators.csrf import csrf_exempt

```
@csrf_exempt
def delete_element(request: HttpRequest):
    if request.method == "DELETE":
        return HttpResponse("")
    return HttpResponseBadRequest("Invalid Request")
```

update_todo_htmx

todo/all_todos_htmx.html

```
{% extends "base.html" %}
{% block body %}
<div id="todo-list">
  {% for todo in todos %}
  <h1 id="todo-{{todo.id}}">
    {{ todo.name }}
    <a hx-get="{% url 'get_todo_htmx' todo.id %}" hx-target="#todo-tooltips">get detail</a>
    <a hx-get="{% url 'update_todo_htmx' todo.id %}" hx-target="#todo-tooltips">update todo</a>
    <a hx-delete="{% url 'delete_todo_htmx' todo.id %}" hx-target="#todo-{{todo.id}}" hx-swap="outerHTML"
      hx-confirm="are you sure you want to delete this todo {{ todo.name }}?">delete todo</a>
  </h1>

  {% endfor %}
</div>

<a hx-get="{% url 'create_todo_htmx' %}" hx-target="#todo-tooltips">create todo v1</a>
<a hx-get="{% url 'create_todo_htmx_v2' %}" hx-target="#todo-tooltips">create todo v2</a>
<h1 id="todos-length">Todos length: {{ count }}</h1>

<div id="todo-tooltips"></div>


{% endblock body %}
```

update_todo_htmx

todo/views.py

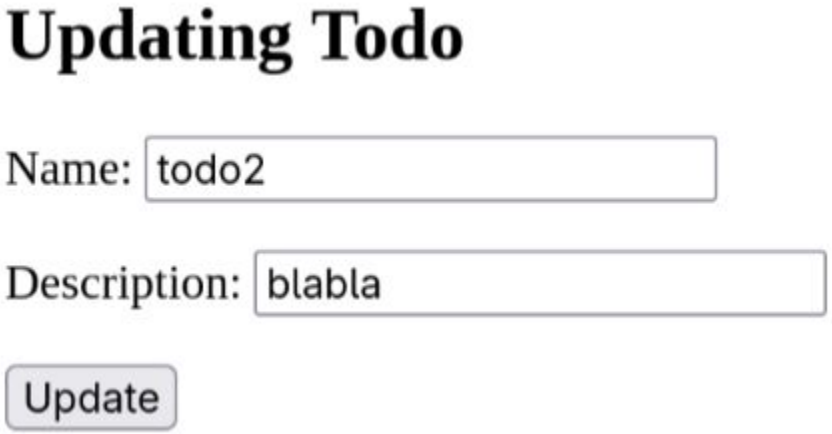
```python
def update_todo_htmx(request: HttpRequest, pk):
    todo = get_object_or_404(Todo, pk=pk)
    if request.method == "POST":
        form = TodoForm(request.POST, instance=todo)
        if form.is_valid():
            todo = form.save()
            return render(
                request, "todo/update_todo_htmx_component.html", {"todo": todo}
            )
    else:
        form = TodoForm(instance=todo)
    return render(request, "todo/update_todo_htmx.html", {"form": form, "todo": todo})
```

update_todo_htmx

todo/update_todo_htmx.html

```
<h2>Updating Todo</h2>

<form hx-post="{% url 'update_todo_htmx' todo.id %}" hx-target="#todo-{{todo.id}}" hx-swap="outerHTML" method="POST">
  {% csrf_token %}
  {{ form.as_p }}
  <button type="submit">Update</button>
</form>
```



update_todo_htmx

todo/update_todo_htmx_component.html

```
<h1 id="todo-{{todo.id}}">
  {{ todo.name }}
  <a hx-get="{% url 'get_todo_htmx' todo.id %}" hx-target="#todo-tooltips">get detail</a>
  <a hx-get="{% url 'update_todo_htmx' todo.id %}" hx-target="#todo-tooltips">update todo</a>
  <a hx-delete="{% url 'delete_todo_htmx' todo.id %}" hx-target="#todo-{{todo.id}}"
    hx-confirm="are you sure you want to delete this todo {{ todo.name }}?">delete todo</a>
</h1>
```

Create_todo_htmx

todo/views.py

```python
def create_todo_htmx(request: HttpRequest):
    if request.method == "POST":
        form = TodoForm(request.POST)
        if form.is_valid():
            todo = form.save()
            return render(request, "todo/new_todo_component.html", {"todo": todo})
    else:
        form = TodoForm()
    return render(request, "todo/create_todo_htmx.html", {"form": form})
```

Create_todo_htmx_v2

todo/views.py

```python
def create_todo_htmx_v2(request: HttpRequest):
    if request.method == "POST":
        form = TodoForm(request.POST)
        if form.is_valid():
            todo = form.save()
            count = len(Todo.objects.all())
            return render(
                request,
                "todo/new_todo_component_v2.html",
                {"todo": todo, "count": count},
            )
    else:
        form = TodoForm()
    return render(request, "todo/create_todo_htmx_v2.html", {"form": form})
```

Create_todo_htmx_v2

todo/create_todo_htmx_v2.py

```
<h2>Create Todo</h2>

<form hx-post="{% url 'create_todo_htmx_v2' %}" hx-target="#todo-list" hx-swap="beforeend" method="post">
  {% csrf_token %}
  {{ form.as_p }}

  <button type="submit">Save</button>
</form>
```

Create_todo_htmx

todo/new_todo_component.html

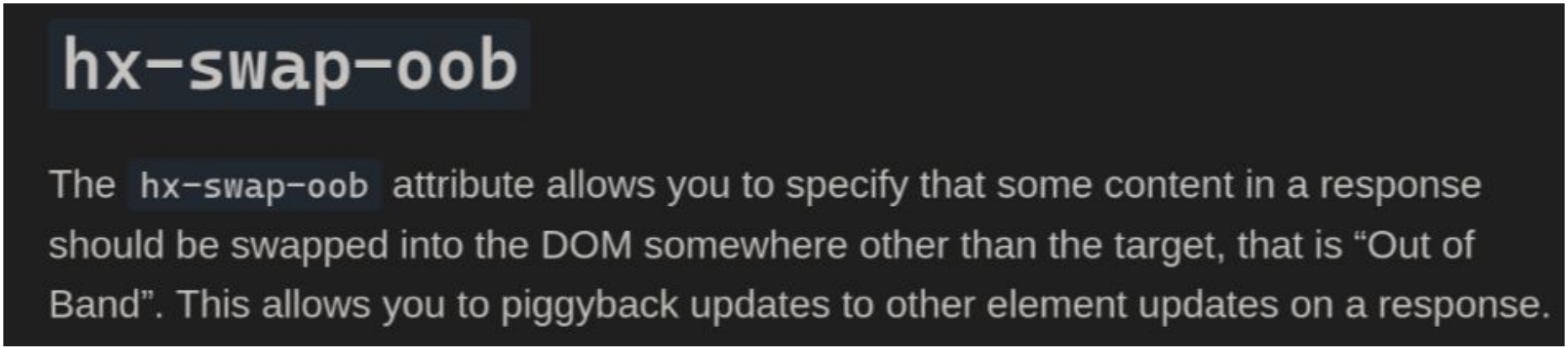
```
<h1 id="todo-{{todo.id}}">
  {{ todo.name }}
  <a hx-get="{% url 'get_todo_htmx' todo.id %}" hx-target="#todo-tooltips">get detail</a>
  <a hx-get="{% url 'update_todo_htmx' todo.id %}" hx-target="#todo-tooltips">update todo</a>
  <a hx-delete="{% url 'delete_todo_htmx' todo.id %}"
    hx-confirm="are you sure you want to delete this todo {{ todo.name }}?">delete todo</a>
</h1>
```

todo/new_todo_component_v2.html

```
<h1 id="todo-{{todo.id}}">
  {{ todo.name }}
  <a hx-get="{% url 'get_todo_htmx' todo.id %}" hx-target="#todo-tooltips">get detail</a>
  <a hx-get="{% url 'update_todo_htmx' todo.id %}" hx-target="#todo-tooltips">update todo</a>
  <a hx-delete="{% url 'delete_todo_htmx' todo.id %}" hx-target="#todo-{{todo.id}}"
    hx-confirm="are you sure you want to delete this todo {{ todo.name }}?">delete todo</a>
</h1>


<h1 id="todos-length" hx-swap-oob="true">Todos length is: {{count}}</h1>
```

hx-swap-oob


hx-swap-oob
The hx-swap-oob attribute allows you to specify that some content in a response should be swapped into the DOM somewhere other than the target, that is “Out of Band”. This allows you to piggyback updates to other element updates on a response.

Consider the following response HTML:

```
<div>
  ...
</div>
<div id="alerts" hx-swap-oob="true">
    Saved!
</div>
```

The first div will be swapped into the target the usual manner. The second div, however, will be swapped in as a replacement for the element with the id `alerts`, and will not end up in the target.

The value of the `hx-swap-oob` can be:

- `true`
- any valid `hx-swap` value
- any valid `hx-swap` value, followed by a colon, followed by a CSS selector

delete_todo_htmx

todo/all_todos_htmx.html

```
{% extends "base.html" %}
{% block body %}
<div id="todo-list">
  {% for todo in todos %}
  <h1 id="todo-{{todo.id}}">
    {{ todo.name }}
    <a hx-get="{% url 'get_todo_htmx' todo.id %}" hx-target="#todo-tooltips">get detail</a>
    <a hx-get="{% url 'update_todo_htmx' todo.id %}" hx-target="#todo-tooltips">update todo</a>
    <a hx-delete="{% url 'delete_todo_htmx' todo.id %}" hx-target="#todo-{{todo.id}}" hx-swap="outerHTML"
      hx-confirm="are you sure you want to delete this todo {{ todo.name }}?">delete todo</a>
  </h1>

  {% endfor %}
</div>

<a hx-get="{% url 'create_todo_htmx' %}" hx-target="#todo-tooltips">create todo v1</a>
<a hx-get="{% url 'create_todo_htmx_v2' %}" hx-target="#todo-tooltips">create todo v2</a>
<h1 id="todos-length">Todos length: {{ count }}</h1>

<div id="todo-tooltips"></div>


{% endblock body %}
```

delete_todo_htmx

todo/views.py

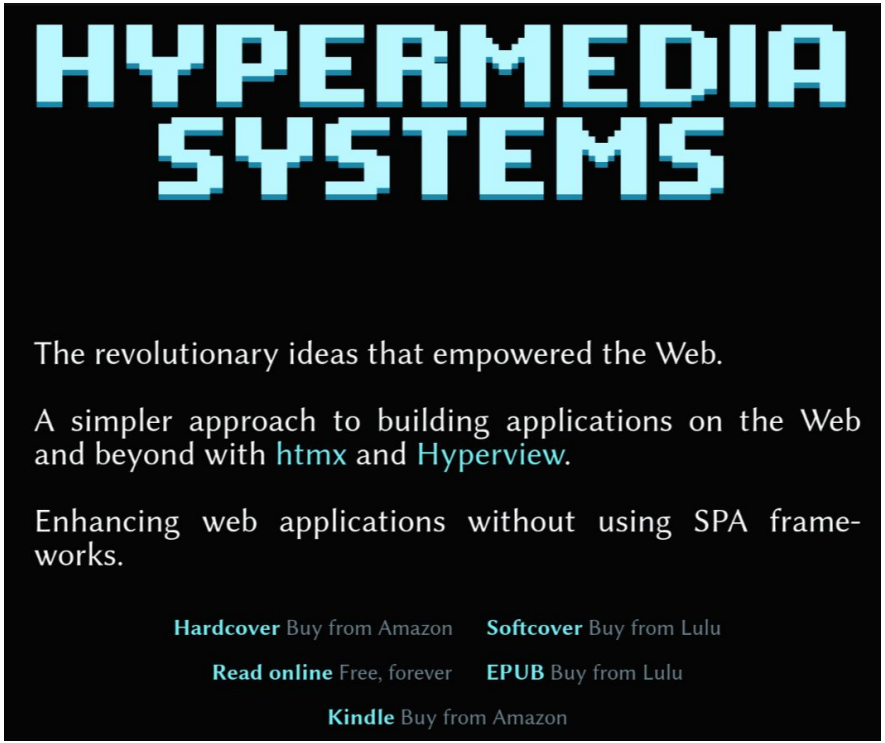
```python
@csrf_exempt
def delete_todo_htmx(request: HttpRequest, pk):
    todo = get_object_or_404(Todo, pk=pk)
    if request.method == "DELETE":
        todo.delete()
        count = len(Todo.objects.all())
        return render(request, "todo/delete_todo_htmx.html", {"count": count})
    return HttpResponseBadRequest("")
```

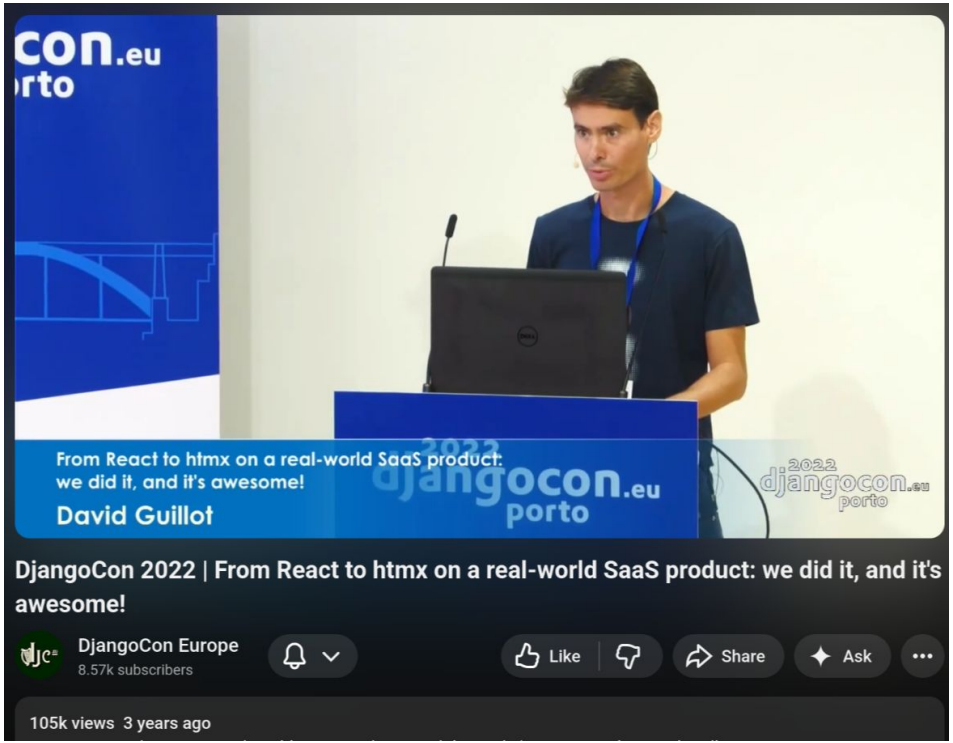
todo/delete_todo_htmx.html

```html
<h1 id="todos-length" hx-swap-oob="true">Todos length is: {{count}}</h1>
```

Whats Next?

https://hypermedia.systems/





Whats Next?

https://django-htmx.readthedocs.io/en/latest/

Welcome to the documentation for django-htmx. This package provides an easy way to include htmx in your Django projects and tools to interact with htmx's HTTP extensions.

THANKS!
Source:
htmx.org
Carson gross
Primeagen