

Lucas Pires

Django + htmx

Patterns for Success

Hello!

 Lucas Pires

 Porto, Portugal

 CheckSec

 5+ years of experience with Django

 3 years of experience with htmx



Why did I embrace htmx?

Simplicity

I've always worked in small teams that need to account for the entire scope of the application regardless of the project's size.

Even a simple SPA can bring a lot of complexity to the project.

Why did I embrace htmx?

I ❤️ Django

Django is an “all batteries included” framework that provides most of the tooling required to build a complete application.

Creating an SPA meant that most of the time I was building functionality from scratch that Django provides (e.g. forms) while leaving the Django ecosystem that I love.

Why did I embrace htmx?

Productivity

htmx allows me to ship applications and features faster, PoC and iterate faster.

Directly interfacing with HTML skips prototyping in Figma or other design tools.

One developer can control both the presentation and business layer avoiding bottlenecks.

An introduction to htmx and what it can offer

00

The Hypermedia approach



htmx can be used to:

Provide small UI/UX improvements to an MPA without custom JavaScript

OR

Implement the look and feel of an SPA without the need of a SPA framework

Hypermedia Driven Applications

Applications that focus on maintaining the state on the server.

If the API provides the operations available for the data at all times, there is (usually) no need for complex UI logic to control these operations.

Hypermedia Driven Applications

Single Page Applications	Hypermedia Driven Applications
Manages business state at all times	Business logic state is maintained on the server
Manages UI state at all times	UI state is managed by simple JS libraries or delegated to plain HTML
API provides only data. Logic required to build the UI.	API provides data and operations, building the UI
Great for high interactivity	May not be the best approach for high interactivity
Requires serialization	Does not require serialization

Take full advantage of Django built-in common patterns

01

Django Views

Create views as on-demand entities

This pattern has also been called:

- “lazy loading”
- “trigger on load”
- “load on demand”

Create endpoints that provide sections and subsections of the UI.
Build views as isolated and single responsibility* entities that can be loaded on demand.

This pattern couples the presentation with the business logic so that it can be reused along the application.

```
hx-target="this"  
hx-trigger="load"
```

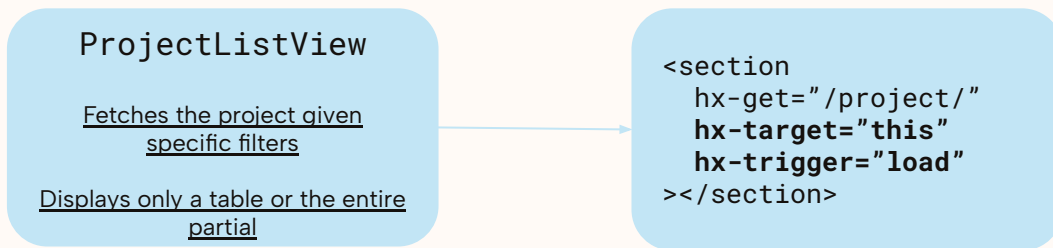
*CRUD is considered here as a single responsibility

Create views as on-demand entities

Building views with this format allows specific sections of the UI to be moved around the application.

Usages can be replicated and configured just by loading the endpoint directly in the HTML.

This ensures the internal behaviour doesn't break regardless of where it is situated on the app.









ProjectListView

Fetches the project given
specific filters

Displays only a table + filters
(partial)

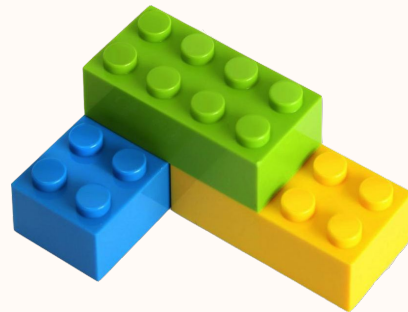
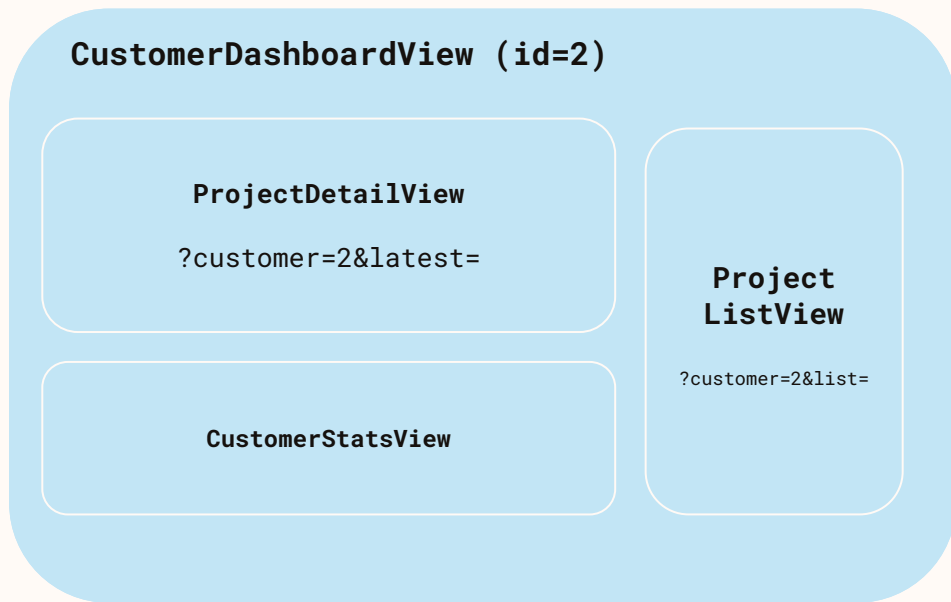
```
<section  
  hx-get="/project/"  
  hx-target="this"  
  hx-trigger="load"  
></section>
```

Title	Customer	Created On	# of Files	Actions
Project A	John Doe	April 16, 2025	0 Files	 
Project B	Phoenix Wright	April 16, 2025	0 Files	 
Project X	John Doe	April 16, 2025	0 Files	 

Create views as on-demand entities

Building a page's UI would follow the composition rule.

Each endpoint has its own responsibility to render each subsection of the page.



```
<div class="flex flex-row gap-2">
  <div class="w-3/4">
    <section
      hx-get="{% url 'project-detail' pk=latest_project_pk %}"
      hx-target="this"
      hx-trigger="load"
    >
  </section>
  <section
    hx-get="{% url 'customer-stats' pk=view.kwargs.pk %}"
    hx-target="this"
    hx-trigger="load"
  >
</section>
</div>
<div class="w-1/4">
  <section
    hx-get="{% url 'project-list' %}"
    hx-vals='{ "customer": {{ view.kwargs.pk }}, "list": "true" }'
    hx-target="this"
    hx-trigger="load"
  >
</section>
</div>
</div>
```

Customer Overview

Project A

Customer: John Doe
Description: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis eu turpis est. Integer vitae fermentum magna. Nulla odio dui, mollis non rhoncus in, iaculis vitae mauris.
Created on: April 16, 2025
0 Files

Project A

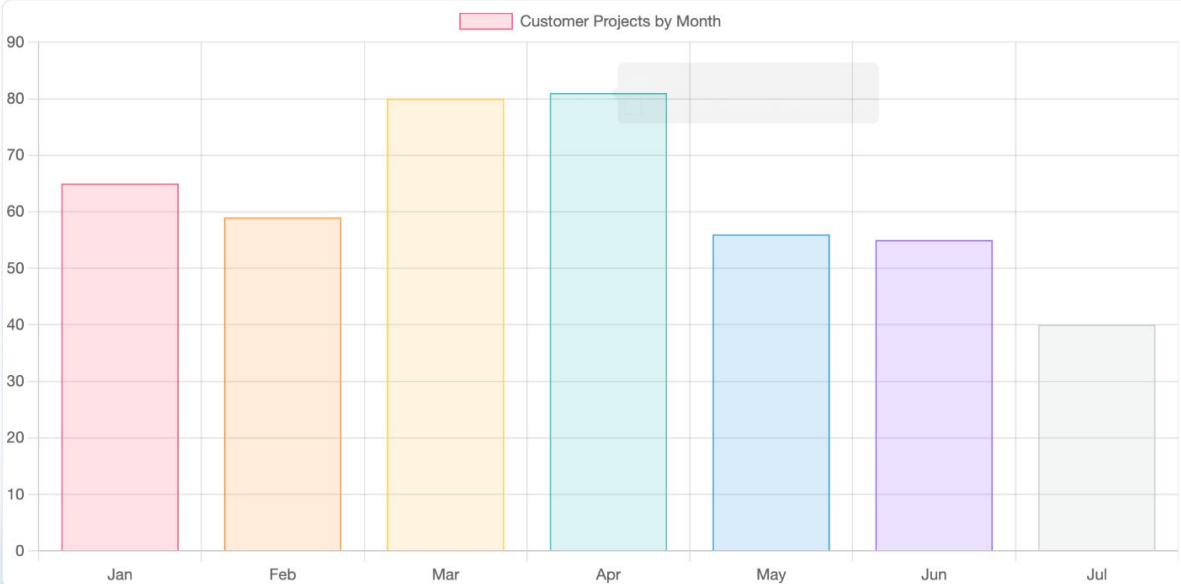
April 16, 2025

Inbox

Project X

April 16, 2025

Inbox



Composition and CBVs

Use the “tried and tested” CBVs internal mixins to provide features to your views.

Not every UI can be created just by using the Django mixins but using them correctly can provide huge productivity boosts.

Create custom mixins to provide functionality that can be easily configured and used across views such as permission checking or context injection.

This moves common patterns to a single point on the codebase to ease testing and ensure that you **DRY**.

ContextMixin
SingleObjectMixin
FormMixin
...

CRUD

Implement all CRUD operations or a subset of them as a single endpoint.
(Similar to DRF ViewSets)

This is possible since htmx unlocks the ability to navigate or submit a form using any HTTP method.

Boilerplate code is reduced.

This can easily be achieved by composing a View using the internal generic mixins or using a package such as **neapolitan**.

Take your templating to the next level!

02

Component Driven Design

Components

Product Item

Wizard Steps

Paginator

User Avatar

Table Filter

Error Toast

Table

Detail
Dropdown

Header



Unitary, reusable and self-contained elements

Composed UI sections with semantic value

Actions
Sidebar

Transactions
Dashboard

Profile Menu

Cart Display

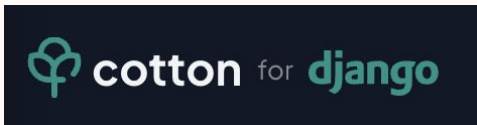
Related
Products

App Navbar

Buyer List

Partials

Component Packages



`simple_block_tag`

`django-bird`

The next step for the Django template language?

django-cotton



- It uses markup similar to HTML
- Supports complex object arguments or strings
- Multiple named slots support
- Dynamic component inclusion
- ...

```
<div id="container">
  <div id="header">
    
    <h1>{{ title }}</h1>
  </div>
  <div id="content">
    {{ slot }}
    {% if price %}
      <div id="price">
        {{ price }}
      </div>
    {% endif %}
  </div>
</div>
```

```
<c-product
  img_url="icon.png"
  title="Item Title"
  price="$10">
  Description of the
  product
</c-product>
```

Component Structure

Having a well-maintained component structure:

- Helps maintaining UI consistency
- Standardizes styling choices
- Maximizes reusability (**DRY**)

Component Structure

One option is to create a two-tiered component structure:

Abstract/Generic Components	Concrete/Specific Components
Provide a basic and extendable skeleton	Created as-is or extended from an abstract component
Work as a layout template	Fills the layout slots with a specific use case
Don't have specific traits or usages	Used for a specific use case
Aren't context specific	Usually bound to a data model or a specific context
e.g. table, badge, menu, toast, dropdown	e.g. user table, status badge, error toast, header menu, product dropdown

A set of abstract components can be one of the building blocks for a design system.







Abstract/Generic Component: Table

```
<table class="w-full text-left table-auto min-w-max">
  <thead>
    <tr>
      {{ head }}
    </tr>
  </thead>
  <tbody>
    {{ body }}
  </tbody>
</table>
```

Concrete/Specific Component: Project Table

```
<c-base.table>
  <c-slot name="head">
    <c-base.th>Title</c-base.th>
    <c-base.th>Customer</c-base.th>
    <c-base.th width="50" end>Actions</c-base.th>
  </c-slot>
  <c-slot name="body">
    {% for project in projects %}
      <tr>
        <c-base.td>{{ project.title }}</c-base.td>
        <c-base.td>{{ project.customer }}</c-base.td>
        <c-base.td>
          <div class="flex gap-3 items-center">
            <a hx-get="{% url 'project-update' pk=project.pk %}" hx-target="main">
              EDIT_ICON
            </a>
          </div>
        </c-base.td>
      </tr>
    {% endfor %}
  </c-slot>
</c-base.table>
```

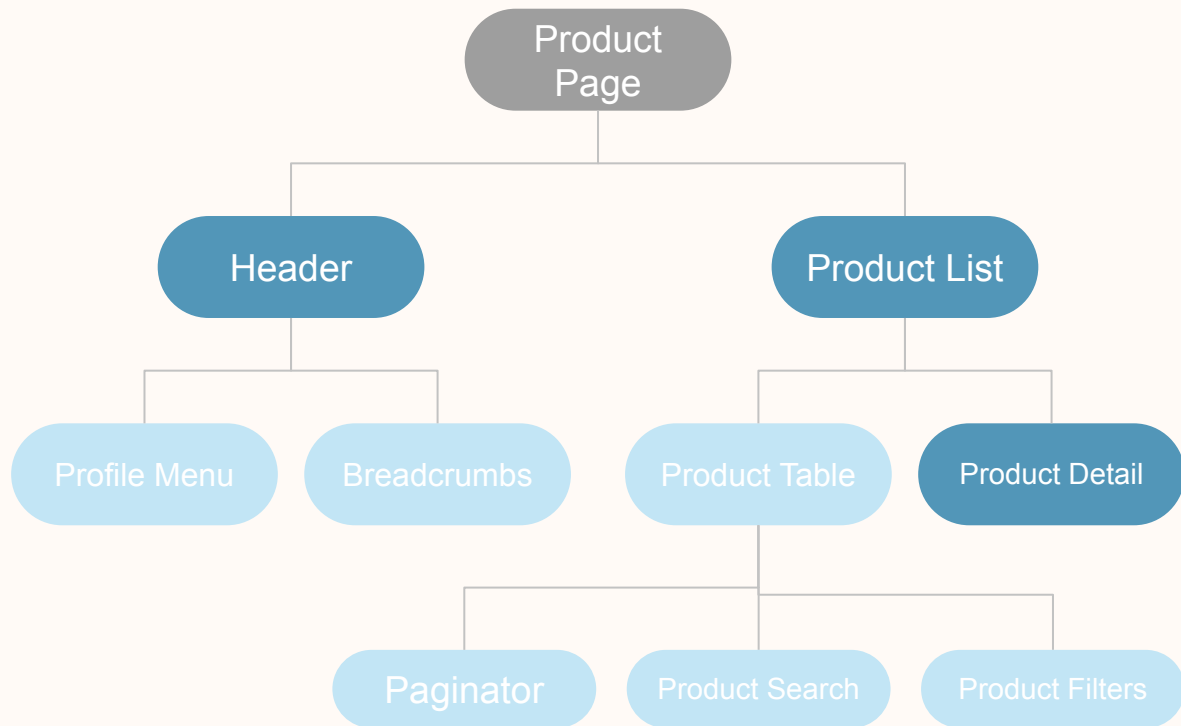
Concrete/Specific Component: Project Table

Title	Customer	Created On	# of Files	Actions
Project A	John Doe	April 16, 2025	0 Files	 
Project B	Phoenix Wright	April 16, 2025	0 Files	 
Project X	John Doe	April 16, 2025	0 Files	 

Page Structure Example

Partial

Component



Use htmx in a way that benefits you and
your project

03

**Be smart about your
htmx**

closest

`hx-target="closest table"`

Use the **closest** keyword on `hx-target`.

Swapping a relative parent is a very common pattern.

It's also less error prone since it avoids complex naming and can be generalized.

Provide feedback to the user

- Make full use of ***hx-indicator*** to show loading indicators or page transitions
- Always display errors by overriding the JS event: ***htmx:responseError*** combined with a simple toast or modal

Use tooling designed to improve DX

- Use *django-htmx* to streamline htmx usage inside your Django codebase
- Use component or partial packages to overcome Django template limitations

Maintaining State in Sync

A very common pattern is to have a page with multiple partials with independent state.

However, certain actions can translate to changes in the state in multiple partials at the same time.

Let's look at the following example:

1. Each outlined partial has a different responsibility: so it's provided by different endpoints

Site Title

LOGG Home About Services Portfol

Sign Sign in

Product A
Product B
Product C
Product D
Product E

Edit Product

Name

Product B

Cancel Submit

2. Submitting the product form and swapping it creates "stale state".

3. The sidebar contains the edited product name but wasn't swapped.

A state dependency was identified.

How to deal with state dependencies? (I)

Extend the swap area

And include the required partials by “lazy loading” them on the template.

Useful:

- If partials are closely related
- If partials can be loaded on demand or rendered without extra business logic
- When dependencies are simple

Another option is to provide the additional business logic on the view and add the required partial for rendering, however that approach breaks the single responsibility principle for each view.

How to deal with state dependencies? (II)

Use Out-Of-Band Swaps

Using this feature, multiple elements contained in the response can be targeted to different DOM locations.

Useful:

- when partials are not directly related
- Secondary elements don't have complex dependencies

How to deal with state dependencies? (III)

Dispatch an event to related partials

By dispatching an event, related partials could sync themselves with the server.

Useful:

- The data is updated on each dependency
- Good for complex dependencies
- More complex to implement, requires event management

Ensure the HTML is easy to work with
since you're going to spend a lot of time
on it!

04

Leverage your HTML

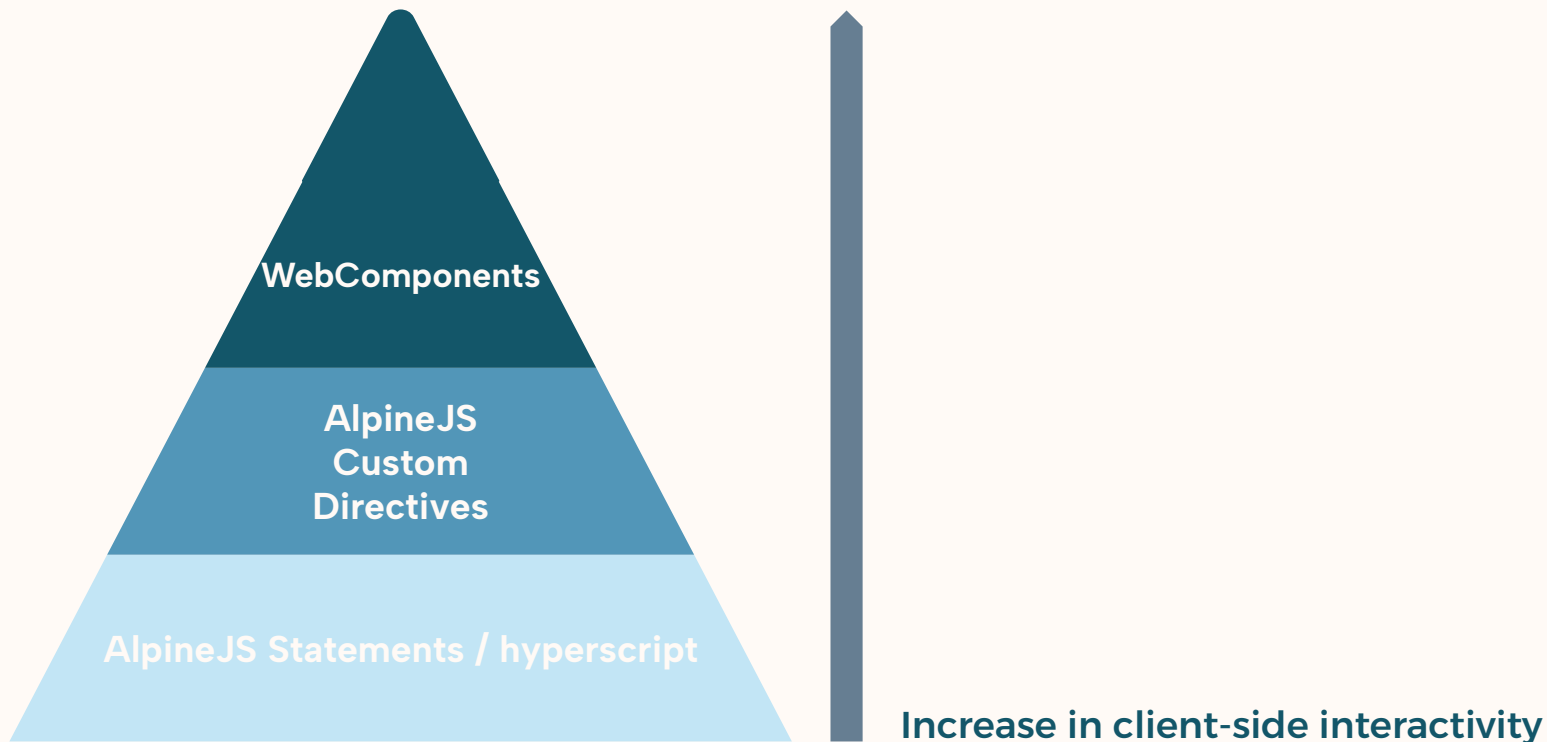
Boost the usage of HTML features

- Use semantic HTML elements to give meaning to specific sections of the page
 - This avoids complex naming of elements and eases when identifying swap targets
- Favor elements such as **<dialog>**, **<details>** and attributes such as **type="search"**, **type="color"** to use native browser features and avoid using 3rd party JavaScript libraries.
- Favor native Web APIs over JavaScript libraries (e.g. Drag and Drop)
- Use the **dataset** property and **data-** attributes to store data.

Scale client-side interactivity accordingly

- Avoid inline scripting
- Use low overhead libraries such as AlpineJS and hyperscript
- Create declarative wrappers with AlpineJS to use external JS libraries
- **Non-trivial client-side interactivity** or **data-oriented reactivity** should be moved to a custom Alpine directive or abstracted using a WebComponent instead of being declared directly on HTML
 - Reduces verbosity on the HTML

Scale client-side interactivity accordingly



LoB

Locality Of Behaviour

- Reduce unnecessary markup verbosity so it's easily understood and maintainable
- Try maintaining the most easy to understand functionality on the markup

Strike a balance between

SoC

Separation of Concerns

- Abstract complex interactivity to its own module
 - An AlpineJS custom directive consolidates behaviour for an entire component in a declarative way
- High reactivity behaviour should be built using proper tooling (React, Svelte) and exported as a WebComponent

Is htmx the productivity boost you need?

05

**Is htmx a good fit for
your project?**

For most cases, yes!

Even more if:

- It's a CRUD heavy application
- Not a lot of need for complex business logic on the client-side
- Not a lot of custom heavy DOM manipulation

However, might not be the best fit for:

- Real-time collaboration
- Applications that depend on a lot of JS libraries to work
- Offline functionality
- Heavy client-side state

The Hybrid Approach: Islands

When complex client-side interactivity is limited to certain sections of the application:

Isolate the sections with complex behaviour into **components** or **islands** built with frontend frameworks.

This can be easily achieved by building UI views and packaging them as **WebComponents**.

Communication between the component can be established via HTML or a JSON API.



What's next for htmx and its peers?

05

The future for hypermedia

Other similar ideas



View Transitions API

- Transitions feel more native
- Make any application look more like an SPA application
- Requires recent browsers versions

View Transition example

[Article 1](#) [Article 2](#)

Article 1 content



photo credit: Michael Kirsh

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam nulla tortor, facilisis vel mauris in, luctus semper turpis. Aliquam lobortis dolor in lacus convallis feugiat. Nulla aliquet ante laoreet enim maximus mollis.

hx-preserve="true"

- Allows DOM elements to be moved without losing state
- May not work on certain elements
- Requires recent browsers

e.g. Moving a YouTube video on the DOM doesn't lose any state

Stability as a Feature

"Websites that are built with jQuery stay online for a very long time, and websites built with htmx should be capable of the same (or better)."

"Going forward, htmx will be developed with its existing users in mind."

"People ... should feel comfortable that the htmx that they write in 2025 will look very similar to htmx they write in 2035 and beyond."

Carson Gross

creator of htmx



Do you have any questions?

Special thanks to:
The amazing team at CheckSec

jlucasp25@gmail.com

Presentation template provided by **Slidesgo**.
Icons by **Flaticon**.
Images by **Freepik**.



Thanks!