django-htmx-admin

Package Design Specification

HTMX-Powered Django Admin Enhancement

Version:	1.0.0 (Alpha)
Date:	November 28, 2025
Target Framework:	Django 4.0+
Dependencies:	django-htmx, htmx.js
License:	MIT

PART 1: FOUNDATIONS — Understanding the Technology

1. What is HTMX?

1.1 The Problem HTMX Solves

Traditional web applications face a fundamental choice: either reload the entire page for every user action (slow, jarring) or build a complex JavaScript frontend (React, Vue, Angular) that requires significant expertise and maintenance overhead.

HTMX provides a third option: extend HTML itself to handle dynamic interactions without writing JavaScript.

1.2 HTMX in Plain English

HTMX is a small JavaScript library (~14KB) that lets you add special attributes to your HTML elements. These attributes tell the browser: "When the user does X, send a request to the server and replace this part of the page with the response."

Example — Traditional vs HTMX Approach:

Traditional (Full Page Reload)	HTMX (Partial Update)
1. User clicks "Delete" 2. Browser sends request 3. Server returns ENTIRE page 4. Browser reloads everything 5. User loses scroll position	1. User clicks "Delete" 2. HTMX sends AJAX request 3. Server returns HTML fragment 4. HTMX swaps just that row 5. Everything else stays put

1.3 Core HTMX Attributes

HTMX works through HTML attributes. Here are the essential ones:

Attribute	Purpose	Example
hx-get	Send GET request when triggered	hx-get="/users/1/edit/"
hx-post	Send POST request (for forms)	hx-post="/users/create/"
hx-delete	Send DELETE request	hx-delete="/users/1/"
hx-target	Where to put the response	hx-target="#user-table"
hx-swap	How to insert the response	hx-swap="outerHTML"
hx-trigger	What event triggers the request	hx-trigger="click"

1.4 The hx-swap Options

Understanding swap modes is critical. They determine HOW the server response replaces content:

Swap Mode	Behavior	Use Case
innerHTML	Replace target's inner content	Update table body
outerHTML	Replace entire target element	Replace a row
beforeend	Insert at end of target	Add new row to table
afterend	Insert after target element	Add sibling element
delete	Delete target element	Remove deleted row

none	Don't swap anything	Trigger events only

2. Django + HTMX: How They Work Together

2.1 The Key Insight

Django already returns HTML. HTMX just asks Django to return smaller pieces of HTML (fragments) instead of full pages. That's it.

Traditional Django Flow:

```
Request → View → Full Template (base.html + content) → Full HTML Page
```

Django + HTMX Flow:

```
HTMX Request → View → Partial Template (fragment only) → HTML Fragment
```

2.2 Detecting HTMX Requests in Django

When HTMX sends a request, it includes a special header. Django can check for this:

```
def my_view(request):
    if request.headers.get('HX-Request'):
        # HTMX request - return just the fragment
        return render(request, 'partials/user_row.html', context)
    else:
        # Normal request - return full page
        return render(request, 'users/list.html', context)
```

With django-htmx package (recommended):

```
def my_view(request):
    if request.htmx:  # Cleaner syntax
        return render(request, 'partials/user_row.html', context)
```

2.3 HX-Trigger: Server-to-Client Communication

HTMX can listen for custom events. Django can trigger these events by including a special response header. This is how we tell the frontend "something happened" without sending HTML.

Example: Triggering a table refresh after deletion:

```
from django.http import HttpResponse

def delete_user(request, pk):
    User.objects.get(pk=pk).delete()
    response = HttpResponse(status=204)  # No content
    response['HX-Trigger'] = 'userDeleted'  # Tell frontend
    return response
```

Frontend listening for the event:

```
<table id="user-table"
       hx-trigger="userDeleted from:body"
       hx-get="/users/"
       hx-swap="innerHTML">
  <!-- Table refreshes when userDeleted event fires -->
</table>
```

2.4 HTTP Status Codes That Matter

HTMX interprets certain status codes specially:

Code	Name	HTMX Behavior
200	OK	Normal swap - replace content with response
204	No Content	Success, no swap - perfect for delete operations
422	Unprocessable Entity	Validation error - still swaps (show form errors)

PART 2: PROJECT OVERVIEW — What We're Building

3. Project Vision

3.1 The Problem with Django Admin

Django Admin is powerful but feels dated. Every action requires a full page reload:

- Editing a single field? Full page reload.
- Deleting a record? Full page reload.
- Filtering the list? Full page reload.
- Changing pages? Full page reload.

This creates a sluggish user experience, especially for admin-heavy applications.

3.2 Our Solution: django-htmx-admin

A drop-in enhancement package that adds HTMX-powered interactions to Django Admin without requiring developers to rewrite their admin classes.

Core Features:

1. **Inline Cell Editing** — Click any cell to edit in place
2. **Modal Forms** — Add/Edit records in popup modals
3. **Instant Filters** — Filter results without page reload
4. **Smooth Deletion** — Delete with confirmation, row fades out
5. **Toast Notifications** — Success/error messages as popups
6. **AJAX Pagination** — Navigate pages without reload

3.3 Usage Example (Developer's Perspective)

After installing our package, a developer can enhance their admin like this:

```
# admin.py
from django.contrib import admin
from htmx_admin import HtmxModelAdmin
from .models import Product

@admin.register(Product)
class ProductAdmin(HtmxModelAdmin):
    list_display = ['name', 'price', 'stock', 'category']

    # New HTMX features:
    list_editable_htmx = ['price', 'stock']
    list_filter_htmx = ['category']
    modal_fields = ['name', 'description', 'price']
```

PART 3: ARCHITECTURE — Technical Design

4. Package Structure

```
django-htmx-admin/
│
├── htmx_admin/
│   ├── __init__.py
│   ├── admin.py              # HtmxModelAdmin class
│   ├── views.py              # AJAX endpoints
│   ├── mixins.py             # Reusable mixins
│   ├── middleware.py         # Toast message handling
│   │
│   ├── templatetags/
│   │   └── htmx_admin_tags.py
│   │
│   ├── templates/htmx_admin/
│   │   ├── change_list.html      # Override admin list
│   │   └── partials/
│   │       ├── table_row.html    # Single row fragment
│   │       ├── edit_cell.html    # Inline edit form
│   │       ├── modal_form.html   # Modal dialog
│   │       └── toast.html        # Notification
│   │
│   └── static/htmx_admin/
│       ├── htmx.min.js
│       ├── htmx-admin.css
│       └── htmx-admin.js
│
├── tests/
├── setup.py
└── README.md
```

5. Module Specifications

5.1 Module A: admin.py — The Main Interface

Developer: A (Backend Focus)

This is the primary class developers will import. It extends Django's ModelAdmin.

Class: HtmxModelAdmin

Attribute	Type	Description
list_editable_htmx	list[str]	Fields that can be edited inline
list_filter_htmx	list[str]	Filters that update without reload
modal_fields	list[str]	Fields to show in modal form
htmx_enabled	bool	Master toggle (default: True)
toast_messages	bool	Show toast notifications (default: True)

Required Method: get_urls()

Must add HTMX-specific URL patterns:

```
def get_urls(self):
    urls = super().get_urls()
    htmx_urls = [
        path(
            '<path:object_id>/htmx-edit/<str:field>/',
            self.admin_site.admin_view(self.htmx_edit_field),
            name='%s_%s_htmx_edit' % info
        ),
        path(
            '<path:object_id>/htmx-delete/',
            self.admin_site.admin_view(self.htmx_delete),
            name='%s_%s_htmx_delete' % info
        ),
        path(
            'htmx-modal/<path:object_id>/',
            self.admin_site.admin_view(self.htmx_modal),
            name='%s_%s_htmx_modal' % info
        ),
    ]
    return htmx_urls + urls
```

Required Method: htmx_edit_field(request, object_id, field)

Handles inline cell editing. GET returns form, POST saves.

```
def htmx_edit_field(self, request, object_id, field):
    obj = self.get_object(request, object_id)

    if request.method == 'GET':
        form = self.get_field_form(obj, field)
        return render(request, 'htmx_admin/partials/edit_cell.html', {
            'form': form,
            'object': obj,
            'field': field
        })

    elif request.method == 'POST':
        form = self.get_field_form(obj, field, data=request.POST)
        if form.is_valid():
            form.save()
            response = render(
                request,
                'htmx_admin/partials/table_cell.html',
                {'object': obj, 'field': field}
            )
            response['HX-Trigger'] = 'cellUpdated'
            return response
        else:
            response = render(
                request,
                'htmx_admin/partials/edit_cell.html',
                {'form': form, 'object': obj, 'field': field}
            )
            response.status_code = 422
            return response
```

Required Method: htmx_delete(request, object_id)

Handles deletion. Returns 204 No Content with HX-Trigger.

```
def htmx_delete(self, request, object_id):
    obj = self.get_object(request, object_id)
    obj_display = str(obj)
    obj.delete()

    response = HttpResponse(status=204)
    response['HX-Trigger'] = json.dumps({
        'rowDeleted': {'id': object_id},
        'showMessage': {
            'level': 'success',
            'message': f'{obj_display} deleted successfully.'
        }
    })
    return response
```

Required Method: htmx_modal(request, object_id)

Modal form for add (object_id='add') or edit operations.

```
def htmx_modal(self, request, object_id):
    if object_id == 'add':
        obj = None
    else:
        obj = self.get_object(request, object_id)

    form_class = self.get_form(request, obj)

    if request.method == 'GET':
        form = form_class(instance=obj)
        return render(
            request,
            'htmx_admin/partials/modal_form.html',
            {'form': form, 'object': obj, 'opts': self.opts}
        )

    elif request.method == 'POST':
        form = form_class(request.POST, request.FILES, instance=obj)
        if form.is_valid():
            self.save_model(request, form.instance, form, change=bool(obj))
            response = HttpResponse(status=204)
            response['HX-Trigger'] = json.dumps({
                'modalClosed': True,
                'tableRefresh': True,
                'showMessage': {
                    'level': 'success',
                    'message': 'Saved successfully.'
                }
            })
            return response
        else:
            response = render(
                request,
                'htmx_admin/partials/modal_form.html',
                {'form': form, 'object': obj, 'opts': self.opts}
            )
            response.status_code = 422
            return response
```

5.2 Module B: mixins.py — Reusable Logic

Developer: A (Backend Focus)

Class: HtmxResponseMixin

```
class HtmxResponseMixin:
    """Mixin providing HTMX response utilities."""

    def is_htmx_request(self, request):
        """Check if request originated from HTMX."""
        return request.headers.get('HX-Request') == 'true'

    def htmx_response(self, content='', status=200, **triggers):
        """Create response with HX-Trigger header."""
        response = HttpResponse(content, status=status)
        if triggers:
            response['HX-Trigger'] = json.dumps(triggers)
        return response

    def htmx_redirect(self, url):
        """Trigger client-side redirect via HTMX."""
        response = HttpResponse(status=204)
        response['HX-Redirect'] = url
        return response
```

Class: HtmxFormMixin

```
class HtmxFormMixin:
    """Mixin for HTMX form handling."""

    def form_invalid(self, form):
        """Return form with 422 status for HTMX to swap."""
        response = super().form_invalid(form)
        response.status_code = 422  # Unprocessable Entity
        return response

    def form_valid(self, form):
        """Save and return success trigger."""
        self.object = form.save()
        response = HttpResponse(status=204)
        response['HX-Trigger'] = json.dumps({
            'formSuccess': True,
            'showMessage': {
                'level': 'success',
                'message': f'{self.object} saved successfully.'
            }
        })
        return response
```

5.3 Module C: middleware.py — Toast Message System

Developer: B (Frontend Focus)

Intercepts Django's messages framework and converts to HTMX triggers.

```
import json
from django.contrib import messages

class HtmxMessageMiddleware:
    """Convert Django messages to HTMX triggers."""

    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        response = self.get_response(request)

        # Only process HTMX requests
        if not request.headers.get('HX-Request'):
            return response

        # Get any pending messages
        storage = messages.get_messages(request)
        if storage:
            message_list = []
            for message in storage:
                message_list.append({
                    'level': message.level_tag,
                    'message': str(message)
                })

            # Merge with existing HX-Trigger
            existing = response.get('HX-Trigger', '{}')
            try:
                triggers = json.loads(existing)
            except json.JSONDecodeError:
                triggers = {}

            triggers['showMessages'] = message_list
            response['HX-Trigger'] = json.dumps(triggers)

        return response
```

5.4 Module D: Templates — Frontend Components

Developer: B (Frontend Focus)

Template: partials/edit_cell.html

```
<form hx-post="{% url
'admin:'|add:opts.app_label|add:'_'|add:opts.model_name|add:'_htmx_edit'
object.pk field %}"
      hx-target="this"
      hx-swap="outerHTML"
      class="htmx-edit-form">
    {% csrf_token %}

    <div class="edit-cell-wrapper">
        {{ form.as_p }}

        <div class="edit-cell-actions">
            <button type="submit" class="btn-save" title="Save">
                ✓
            </button>
            <button type="button"
                    class="btn-cancel"
                    hx-get="..."
                    hx-target="closest .htmx-edit-form"
                    hx-swap="outerHTML"
                    title="Cancel">
                ✕
            </button>
        </div>
    </div>

    {% if form.errors %}
    <div class="edit-cell-errors">{{ form.errors }}</div>
    {% endif %}
</form>
```

Template: partials/modal_form.html

```
<div id="htmx-modal" class="modal-backdrop">
    <div class="modal-dialog">
        <div class="modal-header">
            <h3>
                {% if object %}Edit{% else %}Add{% endif %}
                {{ opts.verbose_name|title }}
            </h3>
            <button type="button" class="modal-close">&times;</button>
        </div>

        <form hx-post="{% url ... %}"
              hx-target="#htmx-modal"
              hx-swap="outerHTML">
            {% csrf_token %}

            <div class="modal-body">
                {{ form.as_div }}
            </div>

            <div class="modal-footer">
                <button type="button" class="btn-cancel">Cancel</button>
```

```
                <button type="submit" class="btn-save">Save</button>
            </div>
        </form>
    </div>
</div>
```

5.5 Module E: htmx-admin.js — Frontend Logic

Developer: B (Frontend Focus)

```
/**
 * django-htmx-admin Frontend Controller
 */
(function() {
    'use strict';

    // ========== Toast Notifications ==========
    function showToast(level, message, duration = 5000) {
        const container = document.getElementById('toast-container');
        const template = document.getElementById('toast-template');

        const toast = template.content.cloneNode(true).firstElementChild;
        toast.className = `toast toast-${level}`;
        toast.querySelector('.toast-message').textContent = message;

        container.appendChild(toast);

        // Auto-dismiss
        setTimeout(() => {
            toast.classList.add('toast-fade-out');
            setTimeout(() => toast.remove(), 300);
        }, duration);
    }

    // Listen for showMessage trigger from server
    document.body.addEventListener('showMessage', function(event) {
        const { level, message } = event.detail;
        showToast(level, message);
    });

    // ========== Modal Management ==========
    document.body.addEventListener('modalClosed', function() {
        const modal = document.getElementById('htmx-modal');
        if (modal) modal.remove();
    });

    // Close modal on Escape key
    document.addEventListener('keydown', function(event) {
        if (event.key === 'Escape') {
            const modal = document.getElementById('htmx-modal');
            if (modal) modal.remove();
        }
    });

    // ========== Row Deletion Animation ==========
    document.body.addEventListener('rowDeleted', function(event) {
        const { id } = event.detail;
        const row = document.querySelector(`[data-row-id="${id}"]`);
        if (row) {
            row.classList.add('row-fade-out');
            setTimeout(() => row.remove(), 300);
        }
    });

})();
```

PART 4: IMPLEMENTATION GUIDE

6. Developer Task Assignment

Developer	Responsibilities	Deliverables
Dev A	Backend / Python • HtmxModelAdmin class • Mixins • URL routing • View logic	admin.py mixins.py views.py urls.py
Dev B	Frontend / Templates • HTML templates • JavaScript handlers • CSS styling • Middleware	All templates/* htmx-admin.js htmx-admin.css middleware.py

7. Implementation Phases

Phase 1: Foundation (Days 1-2)

Goal: Package structure + inline editing working.

Dev A:

1. Create package skeleton with setup.py
2. Implement basic HtmxModelAdmin class
3. Add htmx_edit_field view
4. Configure URL patterns

Dev B:

1. Set up static file structure
2. Create edit_cell.html template
3. Create table_cell.html template
4. Basic CSS for edit form

Phase 2: Core Features (Days 3-5)

Goal: Modal forms, deletion, toast notifications.

Dev A:

1. Implement htmx_modal view (GET and POST)
2. Implement htmx_delete view
3. Add form validation with 422 responses
4. Implement HX-Trigger headers

Dev B:

1. Create modal_form.html with HTMX attributes
2. Create toast notification system
3. Implement HtmxMessageMiddleware
4. Add animations (row fade-out, modal transitions)

Phase 3: Polish & Testing (Days 6-7)

Goal: Tests, documentation, demo app.

Both Developers:

1. Write unit tests for all views
2. Create demo Django project
3. Write README with installation instructions
4. Record demo GIF for GitHub

PART 5: TESTING REQUIREMENTS

8. Required Tests

Package is complete when these tests pass:

Test 1: Inline Edit - Validation Error

```
def test_inline_edit_validation_error(self):
    product = Product.objects.create(name='Test', price=10.00)

    response = self.client.post(
        reverse('admin:shop_product_htmx_edit', args=[product.pk, 'price']),
        data={'price': 'invalid'},
        HTTP_HX_REQUEST='true'
    )

    self.assertEqual(response.status_code, 422)
    self.assertContains(response, 'Enter a number', status_code=422)
```

Test 2: Inline Edit - Success

```
def test_inline_edit_success(self):
    product = Product.objects.create(name='Test', price=10.00)

    response = self.client.post(
        reverse('admin:shop_product_htmx_edit', args=[product.pk, 'price']),
        data={'price': '25.00'},
        HTTP_HX_REQUEST='true'
    )

    self.assertEqual(response.status_code, 200)
    self.assertIn('cellUpdated', response['HX-Trigger'])

    product.refresh_from_db()
    self.assertEqual(product.price, Decimal('25.00'))
```

Test 3: Delete Operation

```
def test_htmx_delete(self):
    product = Product.objects.create(name='Test', price=10.00)
    initial_count = Product.objects.count()

    response = self.client.post(
        reverse('admin:shop_product_htmx_delete', args=[product.pk]),
        HTTP_HX_REQUEST='true'
    )

    self.assertEqual(response.status_code, 204)
    self.assertEqual(Product.objects.count(), initial_count - 1)
    self.assertIn('rowDeleted', response['HX-Trigger'])
```

Test 4: Modal Form - Create

```
def test_modal_create(self):
    initial_count = Product.objects.count()
```

```
    response = self.client.post(
        reverse('admin:shop_product_htmx_modal', args=['add']),
        data={'name': 'New Product', 'price': '99.99'},
        HTTP_HX_REQUEST='true'
    )

    self.assertEqual(response.status_code, 204)
    self.assertEqual(Product.objects.count(), initial_count + 1)
    self.assertIn('tableRefresh', response['HX-Trigger'])
```

APPENDIX: Quick Reference

A. HTMX Attribute Cheat Sheet

Attribute	Description
hx-get="/url"	Send GET request to URL on trigger
hx-post="/url"	Send POST request to URL on trigger
hx-delete="/url"	Send DELETE request to URL on trigger
hx-target="#id"	Element to swap content into (CSS selector)
hx-swap="..."	innerHTML, outerHTML, beforeend, afterend, delete, none
hx-trigger="..."	click, change, submit, keyup, load
hx-confirm="..."	Show confirmation dialog before request

B. Response Header Reference

Header	Purpose
HX-Trigger	Trigger client-side events: {"eventName": data}
HX-Redirect	Client-side redirect to URL
HX-Refresh	Full page refresh (value: "true")

C. Resources

- **HTMX Docs:** https://htmx.org/docs/
- **django-htmx:** https://django-htmx.readthedocs.io/
- **Django Admin:** https://docs.djangoproject.com/en/stable/ref/contrib/admin/

django-htmx-admin — Quick Start Guide

Read the full spec document first! This is just a quick reference.

🎯 What Are We Building?

A Django package that makes Django Admin feel modern by using HTMX for:

- **Inline editing** — Click a cell, edit, save without page reload
- **Modal forms** — Add/edit in popups
- **Instant filters** — Filter without page reload
- **Smooth deletions** — Row fades out
- **Toast notifications** — Success/error popups

🧠 HTMX in 60 Seconds

HTMX = HTML attributes that make AJAX requests.

```
<!-- Before: Link causes full page reload -->
<a href="/delete/1/">Delete</a>

<!-- After: HTMX deletes without reload -->
<button hx-delete="/delete/1/"
        hx-target="closest tr"
        hx-swap="outerHTML"
        hx-confirm="Are you sure?">
    Delete
</button>
```

Key attributes:

Attribute	What it does
`hx-get`	Send GET request
`hx-post`	Send POST request
`hx-delete`	Send DELETE request
`hx-target`	Where to put response
`hx-swap`	How to insert (innerHTML, outerHTML, delete)
`hx-trigger`	What event triggers it (click, change, submit)

🔄 The Django + HTMX Pattern

```
User clicks button with hx-get="/users/1/edit/"
    ↓
HTMX sends AJAX request (with HX-Request header)
    ↓
Django view detects HTMX request
    ↓
Django returns HTML FRAGMENT (not full page)
    ↓
HTMX swaps fragment into hx-target element
```

Django side:

```
def edit_user(request, pk):
    user = User.objects.get(pk=pk)

    if request.headers.get('HX-Request'):
        # HTMX request → return fragment
        return render(request, 'partials/edit_form.html', {'user': user})
    else:
        # Normal request → return full page
        return render(request, 'users/edit.html', {'user': user})
```

📊 HTTP Status Codes That Matter

Code	Name	When to use
200	OK	Normal response with HTML to swap
204	No Content	Success but nothing to swap (delete, modal close)
422	Unprocessable	Validation error (HTMX still swaps to show errors)

📤 Server → Client Communication (HX-Trigger)

Django can trigger JavaScript events by setting a response header:

```
response = HttpResponse(status=204)
response['HX-Trigger'] = json.dumps({
    'rowDeleted': {'id': 123},
    'showMessage': {'level': 'success', 'message': 'Deleted!'}
})
return response
```

Frontend listens:

```
document.body.addEventListener('showMessage', function(event) {
    showToast(event.detail.level, event.detail.message);
});
```

👥 Developer Assignment

Dev A (Backend)

- [ ] `admin.py` — HtmxModelAdmin class
- [ ] `mixins.py` — HtmxResponseMixin, HtmxFormMixin
- [ ] `views.py` — htmx_edit_field, htmx_delete, htmx_modal
- [ ] URL routing

Dev B (Frontend)

- [ ] `templates/partials/*.html` — All HTML fragments
- [ ] `htmx-admin.js` — Toast system, modal handlers
- [ ] `htmx-admin.css` — Styling
- [ ] `middleware.py` — Toast message middleware

✅ Definition of Done

Package is complete when these tests pass:

1. **Inline edit validation error** → Returns 422 with form errors
2. **Inline edit success** → Returns 200, field updated in DB
3. **Delete operation** → Returns 204, object deleted, HX-Trigger set
4. **Modal create** → Returns 204, new object in DB
5. **Toast middleware** → Django messages appear as HX-Trigger

---

📚 Resources

- HTMX Docs
- django-htmx Package
- HTMX Examples

Full specification: `django_htmx_admin_spec.docx`

— End of Specification Document —