TWIST ERP - Backend Admin UI Implementation Guide

Django Admin Interface & Templates

Version: 1.0
Date: October 2025
Framework: Django 4.2+ Admin

1. Django Admin Architecture

1.1 Custom Admin Site

The TWIST ERP admin interface is built on Django's powerful admin framework with extensive customizations for multi-company support, enhanced UI, and business-specific workflows.

Key Features:

- Custom branded admin site
- Company-aware data filtering
- Inline editing for related records
- Bulk actions for common tasks
- Color-coded status indicators
- Advanced search and filtering
- Export to CSV/Excel functionality
- Custom dashboards per module

2. Main Admin Configuration

2.1 Custom Admin Site (core/admin.py)

```
from django.contrib import admin
from django.contrib.admin import AdminSite
from django.utils.translation import gettext_lazy as _
from django.urls import path
from django.shortcuts import render
from django.db.models import Sum, Count
from apps.companies.models import Company

class TwistERPAdminSite(AdminSite):
    site_header = _('TWIST ERP Administration')
```

```python
    site_title = _('TWIST ERP Admin')
    index_title = _('Welcome to TWIST ERP')
    site_url = '/'  # Link to frontend

    def each_context(self, request):
        context = super().each_context(request)

        # Add company selector
        context['companies'] = Company.objects.filter(is_active=True)
        context['current_company'] = getattr(request, 'company', None)

        # Add dashboard stats
        if hasattr(request, 'company'):
            context['stats'] = self.get_dashboard_stats(request.company)

        return context

    def get_dashboard_stats(self, company):
        from apps.finance.models import Invoice
        from apps.sales.models import SalesOrder
        from apps.inventory.models import Product

        return {
            'total_revenue': Invoice.objects.filter(
                company=company,
                invoice_type='AR',
                status='POSTED'
            ).aggregate(total=Sum('total_amount'))['total'] or 0,

            'pending_orders': SalesOrder.objects.filter(
                company=company,
                status='CONFIRMED'
            ).count(),

            'low_stock_products': Product.objects.filter(
                company=company,
                track_inventory=True
            ).count(),  # Add actual stock check

            'pending_approvals': 0,  # Add workflow query
        }

    def get_urls(self):
        urls = super().get_urls()
        custom_urls = [
            path('dashboard/', self.admin_view(self.dashboard_view), name='dashboard'),
        ]
        return custom_urls + urls

    def dashboard_view(self, request):
        context = self.each_context(request)
        context['title'] = 'Dashboard'
        return render(request, 'admin/dashboard.html', context)

# Create custom admin site
admin_site = TwistERPAdminSite(name='twist_admin')
```

2.2 Admin Dashboard Template (templates/admin/dashboard.html)

```
{% extends "admin/base_site.html" %}
{% load i18n static %}

{% block content %}
<div>
    <h1>{% trans "Dashboard" %} - {{ current_company.name }}</h1>


    <div>
        <div>
            <div>□</div>
            <div>
                <div>৳ {{ stats.total_revenue|floatformat:2|intcomma }}</div>
                <div>{% trans "Total Revenue" %}</div>
            </div>
        </div>

        <div>
            <div>□</div>
            <div>
                <div>{{ stats.pending_orders }}</div>
                <div>{% trans "Pending Orders" %}</div>
            </div>
        </div>

        <div>
            <div>□</div>
            <div>
                <div>{{ stats.low_stock_products }}</div>
                <div>{% trans "Low Stock Items" %}</div>
            </div>
        </div>

        <div>
            <div>⚠</div>
            <div>
                <div>{{ stats.pending_approvals }}</div>
                <div>{% trans "Pending Approvals" %}</div>
            </div>
        </div>
    </div>


    <div>
        <h2>{% trans "Recent Activities" %}</h2>
        &lt;table class="activity-table"&gt;
            &lt;thead&gt;
                &lt;tr&gt;
                    &lt;th&gt;{% trans "Time" %}&lt;/th&gt;
                    &lt;th&gt;{% trans "User" %}&lt;/th&gt;
                    &lt;th&gt;{% trans "Action" %}&lt;/th&gt;
                    &lt;th&gt;{% trans "Object" %}&lt;/th&gt;
                &lt;/tr&gt;
            &lt;/thead&gt;
            &lt;tbody&gt;
                {% for activity in recent_activities %}
                &lt;tr&gt;
                    &lt;td&gt;{{ activity.timestamp|timesince }} ago&lt;/td&gt;
                    &lt;td&gt;{{ activity.user }}&lt;/td&gt;
                    &lt;td&gt;{{ activity.action }}&lt;/td&gt;
                    &lt;td&gt;{{ activity.object }}&lt;/td&gt;
                &lt;/tr&gt;
                {% empty %}
                &lt;tr&gt;
                    &lt;td colspan="4"&gt;{% trans "No recent activities" %}&lt;/td&gt;
                &lt;/tr&gt;
                {% endfor %}
            &lt;/tbody&gt;
        &lt;/table&gt;
    </div>
</div>

&lt;style&gt;
.dashboard-container {
    padding: 20px;
}

.dashboard-stats {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
    gap: 20px;
    margin-bottom: 30px;
}

.stat-card {
    background: white;
    border-radius: 8px;
    padding: 20px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
    display: flex;
    align-items: center;
    gap: 15px;
}

.stat-icon {
    font-size: 40px;
    width: 60px;
    height: 60px;
    display: flex;
    align-items: center;
    justify-content: center;
    border-radius: 8px;
}

.stat-icon.revenue { background: #e6f7ff; }
.stat-icon.orders { background: #fff7e6; }
.stat-icon.inventory { background: #f0f5ff; }
.stat-icon.approvals { background: #fff1f0; }

.stat-value {
    font-size: 24px;
    font-weight: bold;
    color: #333;
}

.stat-label {
    color: #666;
    font-size: 14px;
}

.dashboard-section {
    background: white;
    border-radius: 8px;
    padding: 20px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

.activity-table {
    width: 100%;
    border-collapse: collapse;
}

.activity-table th {
    background: #f0f2f5;
    padding: 12px;
    text-align: left;
    font-weight: 600;
}

.activity-table td {
    padding: 12px;
    border-bottom: 1px solid #f0f0f0;
}
&lt;/style&gt;
{% endblock %}
```

3. Finance Module Admin

3.1 Enhanced Account Admin

```
from django.contrib import admin
from django.utils.html import format_html
from django.urls import reverse
from django.utils.safestring import mark_safe
from .models import Account, JournalVoucher, Invoice, Payment

@admin.register(Account)
class AccountAdmin(admin.ModelAdmin):
    list_display = [
        'code', 'name', 'account_type_badge', 'colored_balance',
        'currency', 'is_active_icon', 'action_links'
    ]
    list_filter = [
```

```python
        'account_type', 'is_active', 'is_bank_account',
        'is_control_account', 'company'
    ]
    search_fields = ['code', 'name']
    ordering = ['code']
    list_per_page = 50

    fieldsets = (
        ('Basic Information', {
            'fields': ('company', 'code', 'name', 'account_type'),
            'classes': ('wide',)
        }),
        ('Hierarchy', {
            'fields': ('parent_account',),
        }),
        ('Configuration', {
            'fields': (
                'currency', 'is_bank_account', 'is_control_account',
                'allow_direct_posting'
            ),
            'classes': ('collapse',)
        }),
        ('Status', {
            'fields': ('is_active',),
        }),
    )

    def account_type_badge(self, obj):
        colors = {
            'ASSET': '#52c41a',
            'LIABILITY': '#fa8c16',
            'EQUITY': '#722ed1',
            'REVENUE': '#1890ff',
            'EXPENSE': '#f5222d',
        }
        return format_html(
            '<span>{}</span>',
            colors.get(obj.account_type, '#999'),
            obj.get_account_type_display()
        )
    account_type_badge.short_description = 'Type'
    account_type_badge.admin_order_field = 'account_type'

    def colored_balance(self, obj):
        color = '#52c41a' if obj.current_balance &gt;= 0 else '#f5222d'
        return format_html(
            '<span>৳ {:,.2f}</span>',
            color,
            abs(obj.current_balance)
        )
    colored_balance.short_description = 'Balance'
    colored_balance.admin_order_field = 'current_balance'

    def is_active_icon(self, obj):
        if obj.is_active:
            return format_html(
                '<span>✓</span>'
            )
        return format_html(
            '<span>✗</span>'
        )
    is_active_icon.short_description = 'Active'
    is_active_icon.admin_order_field = 'is_active'

    def action_links(self, obj):
        return format_html(
            '<a href="{}">▯ Ledger</a>'
            '<a href="{}">▯ Report</a>',
            reverse('admin:account_ledger', args=[obj.pk]),
            reverse('admin:account_report', args=[obj.pk])
        )
    action_links.short_description = 'Actions'

    actions = ['activate_accounts', 'deactivate_accounts', 'export_chart']

    def activate_accounts(self, request, queryset):
        updated = queryset.update(is_active=True)
        self.message_user(
            request,
            f'{updated} accounts activated successfully.',
            level='success'
        )
    activate_accounts.short_description = 'Activate selected accounts'

    def deactivate_accounts(self, request, queryset):
        updated = queryset.update(is_active=False)
        self.message_user(
            request,
            f'{updated} accounts deactivated successfully.',
            level='warning'
        )
    deactivate_accounts.short_description = 'Deactivate selected accounts'

    def export_chart(self, request, queryset):
        # Export to Excel
        pass
    export_chart.short_description = 'Export chart of accounts'
```

3.2 Journal Voucher with Inline Entries

```
from django.core.exceptions import ValidationError

class JournalEntryInline(admin.TabularInline):
    model = JournalEntry
    extra = 2
    fields = [
        'line_number', 'account', 'debit_amount',
        'credit_amount', 'description', 'cost_center'
    ]

    def get_formset(self, request, obj=None, **kwargs):
        formset = super().get_formset(request, obj, **kwargs)

        # Custom validation
        original_clean = formset.clean

        def clean_with_balance_check():
            original_clean()

            total_debit = 0
            total_credit = 0

            for form in formset.forms:
                if form.cleaned_data and not form.cleaned_data.get('DELETE'):
                    total_debit += form.cleaned_data.get('debit_amount', 0)
                    total_credit += form.cleaned_data.get('credit_amount', 0)

            if abs(total_debit - total_credit) &gt; 0.01:
                raise ValidationError(
                    f'Journal entries must balance. '
                    f'Debit: {total_debit}, Credit: {total_credit}'
                )

        formset.clean = clean_with_balance_check
        return formset

@admin.register(JournalVoucher)
class JournalVoucherAdmin(admin.ModelAdmin):
    list_display = [
        'voucher_number', 'journal', 'entry_date',
        'status_badge', 'balance_check', 'posted_info'
    ]
    list_filter = ['status', 'journal', 'entry_date', 'company']
    search_fields = ['voucher_number', 'description', 'reference']
    date_hierarchy = 'entry_date'
    inlines = [JournalEntryInline]
    readonly_fields = ['voucher_number', 'posted_by', 'posted_at']

    fieldsets = (
        ('Voucher Information', {
            'fields': (
                'company', 'voucher_number', 'journal',
                'entry_date', 'period'
            )
        }),
        ('Details', {
            'fields': ('reference', 'description')
        }),
        ('Posting Information', {
            'fields': ('status', 'posted_by', 'posted_at'),
            'classes': ('collapse',)
        }),
    )

    def status_badge(self, obj):
        colors = {
            'DRAFT': '#d9d9d9',
            'POSTED': '#52c41a',
            'CANCELLED': '#f5222d'
        }
        return format_html(
            '<span>'
            '{}</span>',
            colors.get(obj.status, '#999'),
            obj.status
        )
    status_badge.short_description = 'Status'

    def balance_check(self, obj):
        entries = obj.entries.all()
        total_debit = sum(e.debit_amount for e in entries)
        total_credit = sum(e.credit_amount for e in entries)

        if abs(total_debit - total_credit) &lt; 0.01:
            return format_html(
                '<span>✓ Balanced</span><br>'
                '&lt;small&gt;Dr: ৳{:,.2f} | Cr: ৳{:,.2f}&lt;/small&gt;',
                total_debit, total_credit
            )
        else:
            return format_html(
                '<span>✗ Unbalanced</span><br>'
                '&lt;small&gt;Diff: ৳{:,.2f}&lt;/small&gt;',
                abs(total_debit - total_credit)
            )
    balance_check.short_description = 'Balance'

    def posted_info(self, obj):
        if obj.status == 'POSTED' and obj.posted_at:
            return format_html(
                '&lt;small&gt;{}<br>by {}&lt;/small&gt;',
                obj.posted_at.strftime('%Y-%m-%d %H:%M'),
                obj.posted_by.username if obj.posted_by else '-'
            )
        return '-'
    posted_info.short_description = 'Posted'

    actions = ['post_vouchers', 'cancel_vouchers']

    def post_vouchers(self, request, queryset):
        from django.utils import timezone

        count = 0
        for voucher in queryset.filter(status='DRAFT'):
            voucher.status = 'POSTED'
            voucher.posted_by = request.user
            voucher.posted_at = timezone.now()
            voucher.save()

            # Post to GL
            voucher.post_to_ledger()
            count += 1
        self.message_user(
            request,
            f'{count} vouchers posted successfully.',
            level='success'
        )
    post_vouchers.short_description = 'Post selected vouchers'
```

4. Inventory Module Admin

4.1 Product Admin with Stock Info

```
@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    list_display = [
        'product_image', 'code', 'name', 'category',
        'colored_cost', 'colored_price', 'margin_percent',
        'stock_badge', 'is_active_icon'
    ]
    list_filter = [
        'product_type', 'category', 'track_inventory',
        'is_active', 'company'
    ]
    search_fields = ['code', 'name', 'description']
    list_editable = ['is_active']
    list_per_page = 50

    fieldsets = (
        ('Basic Information', {
            'fields': (
                'company', 'code', 'name', 'description',
                'category', 'product_type', 'image'
            )
        }),
        ('Unit &amp; Tracking', {
            'fields': (
                'uom', 'track_inventory', 'track_serial',
                'track_batch', 'has_variants'
            )
        }),
        ('Pricing', {
            'fields': ('cost_price', 'selling_price', 'tax_category'),
            'classes': ('wide',)
        }),
        ('Inventory Control', {
            'fields': (
                'reorder_level', 'reorder_quantity',
                'min_order_qty', 'max_order_qty'
            ),
            'classes': ('collapse',)
        }),
        ('Accounting', {
            'fields': (
                'inventory_account', 'income_account',
                'expense_account'
            ),
            'classes': ('collapse',)
        }),
        ('Status', {
            'fields': ('is_active',)
        }),
    )

    def product_image(self, obj):
        if obj.image:
            return format_html(
                '<img>',
                obj.image.url
            )
        return format_html(
            '<div>▯</div>'
        )
    product_image.short_description = ''

    def colored_cost(self, obj):
        return format_html('৳ {:,.2f}', obj.cost_price)
    colored_cost.short_description = 'Cost'
    colored_cost.admin_order_field = 'cost_price'

    def colored_price(self, obj):
        return format_html(
            '<strong>৳ {:,.2f}</strong>',
            obj.selling_price
        )
    colored_price.short_description = 'Price'
    colored_price.admin_order_field = 'selling_price'

    def margin_percent(self, obj):
        if obj.cost_price &gt; 0:
            margin = ((obj.selling_price - obj.cost_price) / obj.cost_price) * 100
            color = '#52c41a' if margin &gt; 20 else '#fa8c16'
            return format_html(
                '<span>{:.1f}%</span>',
                color, margin
            )
        return '-'
    margin_percent.short_description = 'Margin'

    def stock_badge(self, obj):
        if not obj.track_inventory:
            return format_html('<span>N/A</span>')

        # Get current stock (simplified)
        from apps.inventory.models import StockLedger
        stock = StockLedger.objects.filter(
            product=obj, company=obj.company
        ).aggregate(
            total=models.Sum('balance_qty')
        )['total'] or 0
        if stock &lt;= 0:
            badge = format_html(
                '<span>'
                'Out of Stock</span>'
            )
        elif stock &lt;= obj.reorder_level:
            badge = format_html(
                '<span>'
                'Low: {:.0f}</span>',
                stock
            )
        else:
            badge = format_html(
                '<span>'
                'Stock: {:.0f}</span>',
                stock
            )

        return badge
    stock_badge.short_description = 'Stock'

    def is_active_icon(self, obj):
        return format_html(
            '<span>{}</span>',
            '#52c41a' if obj.is_active else '#ccc',
            '✓' if obj.is_active else '✗'
        )
    is_active_icon.short_description = 'Active'

    actions = ['check_stock_levels', 'generate_barcodes', 'export_products']
```

5. Sales & Procurement Admin

5.1 Sales Order Admin

```
class SalesOrderLineInline(admin.TabularInline):
    model = SalesOrderLine
    extra = 1
    fields = [
        'product', 'description', 'quantity',
        'unit_price', 'discount_percent', 'line_total_display'
    ]
    readonly_fields = ['line_total_display']

    def line_total_display(self, obj):
        if obj.pk:
            return format_html('৳ {:,.2f}', obj.line_total)
        return '-'
    line_total_display.short_description = 'Total'

@admin.register(SalesOrder)
class SalesOrderAdmin(admin.ModelAdmin):
    list_display = [
```

```python
        'order_number', 'customer_link', 'order_date',
        'delivery_date', 'colored_total', 'status_badge',
        'fulfillment_progress'
    ]
    list_filter = ['status', 'order_date', 'company']
    search_fields = ['order_number', 'customer__name']
    date_hierarchy = 'order_date'
    inlines = [SalesOrderLineInline]

    def customer_link(self, obj):
        url = reverse('admin:sales_customer_change', args=[obj.customer.pk])
        return format_html('<a href="{}">{}</a>', url, obj.customer.name)
    customer_link.short_description = 'Customer'

    def colored_total(self, obj):
        return format_html(
            '<strong>৳ {:,.2f}</strong>',
            obj.total_amount
        )
    colored_total.short_description = 'Total'

    def status_badge(self, obj):
        colors = {
            'DRAFT': '#d9d9d9',
            'CONFIRMED': '#1890ff',
            'PROCESSING': '#fa8c16',
            'DELIVERED': '#52c41a',
            'CANCELLED': '#f5222d'
        }
        return format_html(
            '<span>{}</span>',
            colors.get(obj.status, '#999'),
            obj.status
        )
    status_badge.short_description = 'Status'

    def fulfillment_progress(self, obj):
        # Calculate delivery progress
        lines = obj.lines.all()
        if not lines:
            return '-'

        total_qty = sum(line.quantity for line in lines)
        delivered_qty = sum(line.delivered_qty for line in lines)

        if total_qty &gt; 0:
            percent = (delivered_qty / total_qty) * 100
            return format_html(
                '<div>'
                '<div></div>'
                '<span>{:.0f}%</span>'
                '</div>',
                percent, percent
            )
        return '-'
    fulfillment_progress.short_description = 'Fulfillment'
```

6. Custom Admin Templates

6.1 Base Site Template (templates/admin/base_site.html)

```
{% extends "admin/base.html" %}
{% load static %}

{% block title %}{% if subtitle %}{{ subtitle }} | {% endif %}{{ title }} | TWIST ERP{% e

{% block branding %}
<h1>
    <a href="{% url 'admin:index' %}">
        <img>
        TWIST ERP
    </a>
</h1>
{% endblock %}

{% block userlinks %}

<div>
    &lt;select id="company-switch" style="padding: 5px 10px; border-radius: 4px;"&gt;
        {% for company in companies %}
        &lt;option value="{{ company.id }}" {% if company == current_company %}selected{%
            {{ company.name }}
        &lt;/option&gt;
        {% endfor %}
    &lt;/select&gt;
</div>

{{ block.super }}
{% endblock %}

{% block extrastyle %}
{{ block.super }}
&lt;style&gt;
    #header {
        background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    }

    #branding h1 a {
        color: white;
    }

    .module h2, .module caption {
        background: #667eea;
    }

    a:link, a:visited {
        color: #667eea;
    }

    .button, input[type=submit], input[type=button], .submit-row input {
        background: #667eea;
        border: none;
    }

    .button:hover, input[type=submit]:hover {
        background: #764ba2;
    }
&lt;/style&gt;
{% endblock %}

{% block extrahead %}
{{ block.super }}
&lt;script&gt;
document.getElementById('company-switch')?.addEventListener('change', function() {
    const companyId = this.value;
    // Set company in session and reload
    fetch('/api/v1/set-company/', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'X-CSRFToken': '{{ csrf_token }}'
        },
        body: JSON.stringify({ company_id: companyId })
    }).then(() =&gt; {
        window.location.reload();
    });
});
&lt;/script&gt;
{% endblock %}
```

7. Admin Actions & Filters

7.1 Custom Admin Filters

```
from django.contrib.admin import SimpleListFilter

class LowStockFilter(SimpleListFilter):
    title = 'Stock Level'
    parameter_name = 'stock_level'

    def lookups(self, request, model_admin):
        return (
            ('low', 'Low Stock'),
            ('out', 'Out of Stock'),
            ('normal', 'Normal'),
        )

    def queryset(self, request, queryset):
        if self.value() == 'low':
            # Products below reorder level
            return queryset.filter(
                track_inventory=True
            )  # Add actual stock query
        if self.value() == 'out':
            # Products with zero stock
            return queryset.filter(
                track_inventory=True
            )  # Add actual stock query

        return queryset

class DateRangeFilter(SimpleListFilter):
    title = 'Date Range'
    parameter_name = 'date_range'

    def lookups(self, request, model_admin):
        return (
            ('today', 'Today'),
            ('week', 'This Week'),
            ('month', 'This Month'),
            ('quarter', 'This Quarter'),
            ('year', 'This Year'),
        )

    def queryset(self, request, queryset):
        from datetime import datetime, timedelta
        from django.utils import timezone

        now = timezone.now()

        if self.value() == 'today':
            return queryset.filter(created_at__date=now.date())

        if self.value() == 'week':
            start = now - timedelta(days=now.weekday())
            return queryset.filter(created_at__gte=start)

        if self.value() == 'month':
            return queryset.filter(
                created_at__year=now.year,
                created_at__month=now.month
            )

        return queryset
```

8. Admin Utilities

8.1 Export to Excel

```
from django.http import HttpResponse
import openpyxl
from openpyxl.styles import Font, PatternFill

def export_to_excel(modeladmin, request, queryset):
    """Export queryset to Excel"""
    wb = openpyxl.Workbook()
    ws = wb.active
    ws.title = modeladmin.model._meta.verbose_name_plural

    # Header
    fields = [f.name for f in modeladmin.model._meta.fields]
    ws.append(fields)

    # Style header
    header_fill = PatternFill(start_color="667eea", end_color="667eea", fill_type="solid"
    header_font = Font(bold=True, color="FFFFFF")

    for cell in ws[1]:
        cell.fill = header_fill
        cell.font = header_font

    # Data
    for obj in queryset:
        row = [getattr(obj, field) for field in fields]
        ws.append(row)

    # Response
    response = HttpResponse(
        content_type='application/vnd.openxmlformats-officedocument.spreadsheetml.sheet'
    )
    response['Content-Disposition'] = f'attachment; filename={modeladmin.model._meta.verb

    wb.save(response)
    return response

export_to_excel.short_description = 'Export to Excel'
```

Document Version: 1.0
Total Admin Interfaces: 15+ modules
Status: Production-ready Django admin customizations