# Sales Management — Django App Scaffold

This document contains a ready-to-use Django app named **sales** inside a project named **sales\_project**. It includes models, forms, views, templates and instructions so you can create, run, and extend a Sales Management application where users can add sales records and view simple sales reports.

## Quick Setup (commands)

# 1. Create virtualenv and activate  
python -m venv env  
source env/bin/activate # on Windows: env\Scripts\activate  
  
# 2. Install Django  
pip install django  
  
# 3. Create project and app (or use files below)  
django-admin startproject sales\_project .  
python manage.py startapp sales  
  
# 4. Add 'sales' to INSTALLED\_APPS in sales\_project/settings.py  
# 5. Make migrations and run  
python manage.py makemigrations  
python manage.py migrate  
python manage.py createsuperuser # optional for admin  
python manage.py runserver

## Project structure (provided)

sales\_project/  
├── manage.py  
├── sales\_project/  
│ ├── \_\_init\_\_.py  
│ ├── settings.py  
│ ├── urls.py  
│ └── wsgi.py  
└── sales/  
 ├── \_\_init\_\_.py  
 ├── admin.py  
 ├── apps.py  
 ├── forms.py  
 ├── models.py  
 ├── urls.py  
 ├── views.py  
 ├── templates/  
 │ └── sales/  
 │ ├── base.html  
 │ ├── sale\_list.html  
 │ ├── sale\_form.html  
 │ └── report.html  
 └── migrations/

## sales/models.py

from django.db import models  
from django.urls import reverse  
  
class Customer(models.Model):  
 name = models.CharField(max\_length=200)  
 email = models.EmailField(blank=True, null=True)  
 phone = models.CharField(max\_length=50, blank=True)  
  
 def \_\_str\_\_(self):  
 return self.name  
  
class Product(models.Model):  
 name = models.CharField(max\_length=200)  
 sku = models.CharField(max\_length=100, blank=True)  
 unit\_price = models.DecimalField(max\_digits=10, decimal\_places=2)  
  
 def \_\_str\_\_(self):  
 return self.name  
  
class Sale(models.Model):  
 date = models.DateField()  
 customer = models.ForeignKey(Customer, on\_delete=models.SET\_NULL, null=True, blank=True)  
 product = models.ForeignKey(Product, on\_delete=models.PROTECT)  
 quantity = models.PositiveIntegerField()  
 unit\_price = models.DecimalField(max\_digits=10, decimal\_places=2)  
 note = models.TextField(blank=True)  
  
 class Meta:  
 ordering = ['-date']  
  
 @property  
 def total(self):  
 return self.quantity \* self.unit\_price  
  
 def \_\_str\_\_(self):  
 return f"{self.product.name} — {self.quantity} @ {self.unit\_price} on {self.date}"  
  
 def get\_absolute\_url(self):  
 return reverse('sales:list')

Notes: - Sale.unit\_price is stored per row so historical prices are preserved even if Product.unit\_price changes.

## sales/admin.py

from django.contrib import admin  
from .models import Customer, Product, Sale  
  
@admin.register(Customer)  
class CustomerAdmin(admin.ModelAdmin):  
 list\_display = ('name', 'email', 'phone')  
 search\_fields = ('name', 'email', 'phone')  
  
@admin.register(Product)  
class ProductAdmin(admin.ModelAdmin):  
 list\_display = ('name', 'sku', 'unit\_price')  
 search\_fields = ('name', 'sku')  
  
@admin.register(Sale)  
class SaleAdmin(admin.ModelAdmin):  
 list\_display = ('date', 'product', 'quantity', 'unit\_price', 'total')  
 list\_filter = ('date', 'product')  
 date\_hierarchy = 'date'

## sales/forms.py

from django import forms  
from .models import Sale  
  
class SaleForm(forms.ModelForm):  
 class Meta:  
 model = Sale  
 fields = ['date', 'customer', 'product', 'quantity', 'unit\_price', 'note']  
 widgets = {  
 'date': forms.DateInput(attrs={'type': 'date'}),  
 'note': forms.Textarea(attrs={'rows': 3}),  
 }

## sales/views.py

from django.shortcuts import render, redirect, get\_object\_or\_404  
from django.db.models import Sum, F, FloatField  
from django.urls import reverse  
from .models import Sale, Product  
from .forms import SaleForm  
from datetime import date  
  
# List all sales  
def sale\_list(request):  
 qs = Sale.objects.select\_related('product', 'customer').all()  
 return render(request, 'sales/sale\_list.html', {'sales': qs})  
  
# Create a new sale  
def sale\_create(request):  
 if request.method == 'POST':  
 form = SaleForm(request.POST)  
 if form.is\_valid():  
 form.save()  
 return redirect('sales:list')  
 else:  
 # Pre-fill unit\_price from product when product provided via GET (optional)  
 form = SaleForm()  
 return render(request, 'sales/sale\_form.html', {'form': form})  
  
# Simple reports page  
def report(request):  
 # filters  
 start = request.GET.get('start')  
 end = request.GET.get('end')  
 sales = Sale.objects.all()  
 if start:  
 sales = sales.filter(date\_\_gte=start)  
 if end:  
 sales = sales.filter(date\_\_lte=end)  
  
 # aggregates  
 total\_revenue = sales.aggregate(total=Sum(F('quantity') \* F('unit\_price'), output\_field=FloatField()))['total'] or 0  
  
 # revenue by product  
 by\_product = (  
 sales.values('product\_\_id', 'product\_\_name')  
 .annotate(revenue=Sum(F('quantity') \* F('unit\_price'), output\_field=FloatField()))  
 .order\_by('-revenue')  
 )  
  
 # monthly totals (simple)  
 from django.db.models.functions import TruncMonth  
 monthly = (  
 sales.annotate(month=TruncMonth('date'))  
 .values('month')  
 .annotate(total=Sum(F('quantity') \* F('unit\_price'), output\_field=FloatField()))  
 .order\_by('month')  
 )  
  
 context = {  
 'total\_revenue': total\_revenue,  
 'by\_product': by\_product,  
 'monthly': monthly,  
 'start': start,  
 'end': end,  
 }  
 return render(request, 'sales/report.html', context)

## sales/urls.py

from django.urls import path  
from . import views  
  
app\_name = 'sales'  
  
urlpatterns = [  
 path('', views.sale\_list, name='list'),  
 path('add/', views.sale\_create, name='add'),  
 path('report/', views.report, name='report'),  
]

## sales\_project/urls.py (project-level)

from django.contrib import admin  
from django.urls import path, include  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('', include('sales.urls', namespace='sales')),  
]

## sales/templates/sales/base.html

<!doctype html>  
<html lang="en">  
<head>  
 <meta charset="utf-8">  
 <meta name="viewport" content="width=device-width,initial-scale=1">  
 <title>Sales Management</title>  
 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/modern-css-reset/dist/reset.min.css">  
 <style>  
 body{font-family: system-ui, Arial; padding: 20px; max-width:1100px; margin:0 auto}  
 header{display:flex; gap:12px; align-items:center; margin-bottom:18px}  
 nav a{margin-right:12px}  
 table{width:100%; border-collapse:collapse}  
 th,td{padding:8px; border:1px solid #ddd}  
 .muted{color:#666; font-size:.9rem}  
 </style>  
</head>  
<body>  
 <header>  
 <h1>Sales Management</h1>  
 <nav>  
 <a href="{% url 'sales:list' %}">Sales</a>  
 <a href="{% url 'sales:add' %}">Add Sale</a>  
 <a href="{% url 'sales:report' %}">Reports</a>  
 <a href="/admin/">Admin</a>  
 </nav>  
 </header>  
 <main>  
 {% block content %}{% endblock %}  
 </main>  
</body>  
</html>

## sales/templates/sales/sale\_list.html

{% extends 'sales/base.html' %}  
{% block content %}  
<h2>Sales</h2>  
{% if sales %}  
<table>  
 <thead>  
 <tr><th>Date</th><th>Product</th><th>Customer</th><th>Qty</th><th>Unit</th><th>Total</th><th>Note</th></tr>  
 </thead>  
 <tbody>  
 {% for s in sales %}  
 <tr>  
 <td>{{ s.date }}</td>  
 <td>{{ s.product.name }}</td>  
 <td>{{ s.customer.name if s.customer else '' }}</td>  
 <td>{{ s.quantity }}</td>  
 <td>{{ s.unit\_price }}</td>  
 <td>{{ s.total }}</td>  
 <td class="muted">{{ s.note|default\_if\_none:'' }}</td>  
 </tr>  
 {% endfor %}  
 </tbody>  
</table>  
{% else %}  
<p>No sales yet. <a href="{% url 'sales:add' %}">Add the first sale</a>.</p>  
{% endif %}  
{% endblock %}

## sales/templates/sales/sale\_form.html

{% extends 'sales/base.html' %}  
{% block content %}  
 <h2>Add sale</h2>  
 <form method="post">  
 {% csrf\_token %}  
 <table>  
 {{ form.as\_table }}  
 </table>  
 <button type="submit">Save</button>  
 </form>  
{% endblock %}

## sales/templates/sales/report.html

{% extends 'sales/base.html' %}  
{% block content %}  
 <h2>Reports</h2>  
 <form method="get" style="margin-bottom:12px">  
 <label>Start: <input type="date" name="start" value="{{ start }}"></label>  
 <label>End: <input type="date" name="end" value="{{ end }}"></label>  
 <button type="submit">Filter</button>  
 </form>  
  
 <h3>Total revenue: {{ total\_revenue|floatformat:2 }}</h3>  
  
 <h4>Revenue by product</h4>  
 <table>  
 <thead><tr><th>Product</th><th>Revenue</th></tr></thead>  
 <tbody>  
 {% for p in by\_product %}  
 <tr>  
 <td>{{ p.product\_\_name }}</td>  
 <td>{{ p.revenue|floatformat:2 }}</td>  
 </tr>  
 {% empty %}  
 <tr><td colspan="2">No data</td></tr>  
 {% endfor %}  
 </tbody>  
 </table>  
  
 <h4>Monthly totals</h4>  
 <table>  
 <thead><tr><th>Month</th><th>Total</th></tr></thead>  
 <tbody>  
 {% for m in monthly %}  
 <tr>  
 <td>{{ m.month|date:'Y-m' }}</td>  
 <td>{{ m.total|floatformat:2 }}</td>  
 </tr>  
 {% empty %}  
 <tr><td colspan="2">No data</td></tr>  
 {% endfor %}  
 </tbody>  
 </table>  
{% endblock %}

## sales\_project/settings.py — important additions

Make sure to add 'sales' to INSTALLED\_APPS.

Also (for simplicity) ensure templates DIRs include BASE\_DIR / 'templates' or app template loading is enabled (default is fine). For static files in development, default is OK.

If you want to allow easier decimal math in Sqlite, no change required. Use TIME\_ZONE as appropriate for your region.

## Extras & Next steps

* Improve UX: add JS to auto-fill unit\_price when product selected.
* Add validation to SaleForm to ensure unit\_price is positive and not zero.
* Add edit/delete views for sales.
* Add CSV export for reports and/or charts.
* Add authentication: require login to add/view sales.
* Add pagination to sale\_list.

## Example data (optional fixtures)

You can create some products and customers in the admin panel or create a loaddata fixture. Example via shell:

python manage.py shell

from sales.models import Product, Customer, Sale  
p = Product.objects.create(name='Widget A', sku='W-A', unit\_price=120.00)  
c = Customer.objects.create(name='ABC Ltd', email='contact@abc.com')  
Sale.objects.create(date='2025-09-01', product=p, quantity=3, unit\_price=120.00, customer=c)

If you’d like, I can: - generate full file contents as individual files ready to download, or - add authentication and restrict pages to logged-in users, or - create CSV export and charts on the reports page.

Tell me which extras you want and I’ll add them.