Tops Technology

Module 16)
Python DB and Framework

Presented By : Nandni Vala

Customizing the Django Admin Panel

1.Techniques for customizing the Django admin panel.

- Customizing the Django admin panel allows developers to tailor it to specific needs, enhance usability, and present data effectively.
- **Register Models with Custom Options**
- Customize how models are displayed in the admin panel by defining an admin class and registering it.
- from django.contrib import admin
- from .models import Product
- class ProductAdmin(admin.ModelAdmin):
- list_display = ('name', 'price', 'stock', 'created_at') # Columns to display
- list_filter = ('category', 'created_at') # Add filters
- search_fields = ('name', 'description') # Enable search
- ordering = ('-created_at',) # Default ordering
- admin.site.register(Product, ProductAdmin)

- **Customize Form Layouts**
- Use fields, fieldsets, or custom forms to control the layout of model fields.
- **Using fields**:
- class ProductAdmin(admin.ModelAdmin):
- fields = ('name', 'price', 'category') # Specify displayed fields
- **Using fieldsets**:
- class ProductAdmin(admin.ModelAdmin):
- fieldsets = (
- ('Basic Info', {'fields': ('name', 'price')}),
- ('Category Info', {'fields': ('category',)}),
-)
- **Inline Editing for Related Models**
- Display and edit related models inline using `TabularInline` or `StackedInline`
- from .models import Order, OrderItem
- class OrderItemInline(admin.TabularInline): # Use StackedInline for vertical layout
- model = OrderItem
- extra = 1

- class OrderAdmin(admin.ModelAdmin):
- inlines = [OrderItemInline]
- admin.site.register(Order, OrderAdmin)
- **Add Custom Actions**
- Define actions to perform bulk operations on selected records.
- def mark_as_sold_out(modeladmin, request, queryset):
- queryset.update(stock=0) # Bulk update
- mark_as_sold_out.short_description = "Mark selected items as sold out"
- class ProductAdmin(admin.ModelAdmin):
- actions = [mark_as_sold_out]
- admin.site.register(Product, ProductAdmin)
- **Customize List Display**
- **Formatting Fields**: Define custom methods for formatted or calculated fields.

- class ProductAdmin(admin.ModelAdmin):
- list_display = ('name', 'formatted_price')
- def formatted_price(self, obj):
- return f"${obj.price:.2f}"
- formatted_price.short_description = 'Price'
- **Add Links**: Make fields clickable.
- list_display_links = ('name',)
- **Use Custom Forms**
- Override the default form to add validation or widgets.
- from django import forms
- class ProductForm(forms.ModelForm):
- class Meta:
- model = Product
- fields = '__all__'
- widgets = {
- 'description': forms.Textarea(attrs={'rows': 4}),
- }
- class ProductAdmin(admin.ModelAdmin):
- form = ProductForm
- admin.site.register(Product, ProductAdmin)

- **Extend Admin Templates**
- Customize the appearance of the admin panel by overriding templates.
- Create a directory: templates/admin.
- Override specific templates, such as change_form.html or base_site.html.
- Example: Change site title in base_site.html:
- {% extends "admin/base.html" %}
- {% block title %}My Custom Admin{% endblock %}
- **Add Custom Views**
- Include custom views for specific tasks or reports.
- from django.urls import path
- from django.http import HttpResponse
- from django.template.response import TemplateResponse
- class ProductAdmin(admin.ModelAdmin):
- def get_urls(self):
- urls = super().get_urls()
- custom_urls = [
- path('report/', self.admin_site.admin_view(self.report_view))
-]
- return custom_urls + urls
- def report_view(self, request):
- return TemplateResponse(request, 'admin/report.html', {'data': 'Report Data'})
- admin.site.register(Product, ProductAdmin)

- **Add Custom Branding**
- Modify the `AdminSite` class to customize branding, titles, or logos.
- from django.contrib.admin import AdminSite
- class CustomAdminSite(AdminSite):
- site_header = "My E-Commerce Admin"
- site_title = "Admin Dashboard"
- index_title = "Welcome to the Admin Panel"
- admin_site = CustomAdminSite(name='custom_admin')
- admin_site.register(Product, ProductAdmin)
- **Third-Party Libraries**
- Use libraries like **django-grappelli** or **django-suit** for advanced admin UI enhancements.