Tops Technology

Module 16)
Python DB and Framework

Presented By : Nandni Vala

Django Admin Panel

1.Introduction to Django’s built-in admin panel.

- Django's **built-in admin panel** is a powerful feature that provides a ready-to-use, web-based interface for managing your application's data.
- **Key Features of Django Admin Panel**
- **Automatic Interface**:
- Django generates a complete CRUD (Create, Read, Update, Delete) interface for your models without requiring custom coding.
- **Customizable**:
- You can tailor the admin panel to fit your application's needs by customizing model display, adding filters, and creating custom forms or actions.
- **Secure**:
- Access to the admin panel is restricted to authorized users with staff or superuser status.
- It uses Django's robust authentication and permissions framework.
- **Search and Filtering**:
- Built-in tools for searching and filtering data make it easy to manage large datasets.

- **Integrated with Django Models**:
- The admin panel works directly with the models defined in your application, eliminating the need for separate configuration.
- **Setting Up Django Admin Panel**
- **Create a Superuser**:
- To access the admin panel, you need a superuser account. Use the following command to create one:
  - python manage.py createsuperuser
- Enter the username, email, and password when prompted.
- **Register Models in Admin**:
- For your models to appear in the admin panel, you need to register them in your app's admin.py file.
- Example:
- from django.contrib import admin
- from .models import Product
  - @admin.register(Product)
  - class ProductAdmin(admin.ModelAdmin):
  -     list_display = ('name', 'price', 'stock') # Columns to display in the admin list view
  -     search_fields = ('name',) # Fields searchable in the admin
  -     list_filter = ('category',) # Filters for narrowing down data

- **Access the Admin Panel**:
- Run your development server:
- python manage.py runserver
- **Customizing the Admin Panel**
- **Customizing List Display**:
- Specify which fields of your model should be displayed in the admin list view using the list_display attribute.
- **Adding Search Functionality**:
- Use search_fields to enable a search box for specific fields.
- **Adding Filters**:
- Use list_filter to add filters for narrowing down displayed records.
- **Customizing Forms**:
- Use form or formfield_overrides to customize the forms used in the admin panel.

- **Creating Custom Actions**:
- Add custom bulk actions to the admin interface.
- Example:
- def mark_as_published(modeladmin, request, queryset):
-     queryset.update(status='Published')
- mark_as_published.short_description = "Mark selected items as Published"
- @admin.register(Product)
- class ProductAdmin(admin.ModelAdmin):
-     actions = [mark_as_published]
- **Advantages of Django Admin Panel**
- **Rapid Development**:
- Speeds up the development process by providing an instant interface for data management.
- **No Extra Setup**:
- Works out of the box with minimal configuration.

2.Customizing the Django admin interface to manage database records.

- **Steps to Customize the Django Admin Interface**
- **1. Register Your Models**
- To make a model accessible in the admin interface, register it in the admin.py file of your app. You can use the admin.register decorator or the admin.site.register() method.
- Example:
- from django.contrib import admin
- from .models import Product
- @admin.register(Product)
- class ProductAdmin(admin.ModelAdmin):
-     Pass
- **Customize the List View**
- The list view displays records from the database in a tabular format. You can control what columns appear and add search, filters, and pagination.

- **list_display**: Specify fields to display in the list view.
- **list_editable**: Allow certain fields to be edited directly in the list view.
- **list_filter**: Add filters to the sidebar for narrowing data.
- **search_fields**: Enable a search bar to search specific fields.
- Example:

```
@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    list_display = ('name', 'price', 'stock', 'category')
    list_editable = ('price', 'stock')
    list_filter = ('category', 'is_available')
    search_fields = ('name', 'description')
```

- **Add Inline Models**
- Inline models allow you to manage related records directly from the parent model’s admin interface.

- Example:
- from .models import Product, ProductImage
- class ProductImageInline(admin.TabularInline): # or admin.StackedInline
-     model = ProductImage
-     extra = 1 # Number of empty forms to display
- @admin.register(Product)
- class ProductAdmin(admin.ModelAdmin):
-     inlines = [ProductImageInline]
- **Add Custom Actions**
- Custom actions enable batch operations on selected records in the list view.
- **Use Custom Admin Templates**
- You can override default admin templates to change the layout or appearance. Place your custom templates in an admin directory under your app’s templates directory.
- **Extend Admin Site Settings**
- You can customize the global admin site settings, such as its title and header.