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

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

Django’s built-in admin panel is one of its most powerful features. It provides a ready-to-use interface for managing your application’s data, allowing you to perform CRUD (Create, Read, Update, Delete) operations with ease. Here’s an introduction to its key aspects:

**Key Features**

1. **User-Friendly Interface**: The admin panel is a web-based interface that is intuitive and easy to navigate.
2. **Customization**: Highly customizable to fit the needs of your application. You can customize the appearance and behavior of the admin interface.
3. **Access Control**: Built-in user authentication and permissions system to control who can access the admin panel and what actions they can perform.
4. **Automatic Admin Interface**: Django automatically generates the admin interface for your models.

**Setting Up the Admin Panel**

1. **Create a Superuser**: To access the admin panel, you need a superuser account. You can create one using:

python manage.py createsuperuser

1. **Register Models**: To make a model accessible in the admin panel, you need to register it in admin.py of your app.

* python

# myapp/admin.py

from django.contrib import admin

from .models import Car

admin.site.register(Car)

1. **Access the Admin Panel**: Once your superuser is created and your models are registered, you can access the admin panel by navigating to /admin/ in your web browser.

• Customizing the Django admin interface to manage database records.

 **ModelAdmin Class**: You can customize the admin interface for each model by creating a ModelAdmin class.

* python

# myapp/admin.py

from django.contrib import admin

from .models import Car

class CarAdmin(admin.ModelAdmin):

list\_display = ('make', 'model', 'year')

search\_fields = ('make', 'model')

list\_filter = ('year',)

admin.site.register(Car, CarAdmin)

 **List Display and Filters**: Customize how your data is displayed in the list view and add filters for easy navigation.

 **Inline Models**: You can manage related models directly from the parent model’s admin interface using inlines.

* python

from django.contrib import admin

from .models import Car, Owner

class OwnerInline(admin.TabularInline):

model = Owner

class CarAdmin(admin.ModelAdmin):

inlines = [

OwnerInline,

]

admin.site.register(Car, CarAdmin)