6.Django Models & Admin Interface

Django models are at the heart of Django's Object-Relational Mapping (ORM) system, allowing you to define your data models in Python classes.

Each model class represents a database table, and each attribute of the class represents a database field.

Here's a basic example of a Django model:

```
from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=100)
    age = models.IntegerField()
    email = models.EmailField()
```

In this example:

Person is the name of the model.

name, age, and email are fields of the model.

CharField, IntegerField, and EmailField are examples of field types provided by Django.

Here are some commonly used field types in Django models:

- CharField: Used to store small to medium-sized strings.
- IntegerField: Used for storing integers.
- FloatField: Used for storing floating-point numbers.
- BooleanField: Used for storing True/False values.
- DateField: Used for storing dates.
- DateTimeField: Used for storing date and time.
- TextField: Used for storing large text.
- EmailField: Used for storing email addresses.
- ForeignKey: Used to define a many-to-one relationship between two models.
- ManyToManyField: Used to define a many-to-many relationship between two models.

You can also define custom fields if Django's built-in fields don't meet your needs.

These fields can have various arguments and options to customize their behavior.

For example, max_length for CharField or null and blank to specify whether a field can be empty in the database.

Django's models make it easy to work with databases by abstracting away much of the complexity of SQL, allowing you to interact with your data using Python objects and methods.

Steps to create Django Models &

Step 1: Define Models

In your blog/models.py file, define your models:

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title

class Comment(models.Model):
    post = models.ForeignKey(Post, on_delete=models.CASCADE)
    author = models.CharField(max_length=50)
```

```
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Comment by {self.author} on {self.post.title}"
```

Step 2: Register Models with Admin Interface

Now, let's register these models with the admin interface. In your blog/admin.py file:

```
from django.contrib import admin
from .models import Post, Comment

admin.site.register(Post)
admin.site.register(Comment)
```

Step 3: Create Superuser

Before accessing the admin interface, you need to create a superuser account.

Run the following command in your terminal:

```
python manage.py createsuperuser
```

Step 4: Run Server and Access Admin Interface

Open your browser and navigate to http://127.0.0.1:8000/admin.

Log in with the superuser credentials you created earlier.

You should see the Django admin interface.

Customizing the Admin Interface

You can further customize the admin interface by creating ModelAdmin classes in blog/admin.py. For example, you can customize the display fields, list filters, search fields, and more.

```
from django.contrib import admin
from .models import Post, Comment

class PostAdmin(admin.ModelAdmin):
    list_display = ('title', 'created_at')
    search_fields = ('title', 'content')
```

```
class CommentAdmin(admin.ModelAdmin):
    list_display = ('author', 'post', 'created_at')
    list_filter = ('post',)
    search_fields = ('author', 'content')

admin.site.register(Post, PostAdmin)
admin.site.register(Comment, CommentAdmin)
```

- **Migration commands-wheneever any new class will added in the model ,then migration command should be run or updated.Below we mention the migrations command.**

  python manage.py makemigrations appname

  python manage.py migrate