Django

Django is a high-level Python web framework that helps you build web applications quickly and with clean, reusable code.

- It follows the MVT pattern (Model-View-Template).
- Comes with batteries included – meaning you get authentication, admin panel, database ORM, routing, etc., out of the box.

Example Use Cases : Blogs, eCommerce websites, APIs and admin dashboards...

Django Versions

You can check Django versions here: https://www.djangoproject.com/download/

- There are regular releases and LTS (Long-Term Support) releases.
- Current version (as of 2025): likely 5.x (Check the official link for the latest)

Step-by-Step: Install and Run a Django App

Step 1: Create a project folder

```
mkdir my_django_project
cd my_django_project
```

Step 2: Create a virtual environment

A virtual environment is an isolated space for Python projects. It allows you to install packages specific to a project without affecting the global Python installation.

- Different projects can use different versions of Django or other packages.
- Avoids package conflicts.
- Keeps your project clean and reproducible.

```
python -m venv venv
```

Step 3: Activate the virtual environment

```
venv\Scripts\activate
```

Step 4: Install Django

check Django versions here: https://www.djangoproject.com/download/

```
py -m pip install Django==5.2.1
```

Step 5: Create a Django Project

```
django-admin startproject mysite .
```

Step 6: Run the Development Server

```
python manage.py runserver

<!-- You should see output like: -->
Starting development server at http://127.0.0.1:8000/
```

Step 7: Create a Django App

```
python manage.py startapp blog
```

Step 8: Register the App

Open mysite/settings.py, and add 'blog', to the INSTALLED_APPS list:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    ...
    'blog',  # Add this line
]
```

Step 9: Run Migrations

```
python manage.py migrate
```

This sets up your database tables.

Step 10: Create Superuser (for admin panel)

```
python manage.py createsuperuser
```

Step 11: Run the Server Again

```
python manage.py runserver

<!--  -->
http://127.0.0.1:8000/ → Django homepage
http://127.0.0.1:8000/admin/ → Django admin login (use your superuser credentials)
```

What is a Django Project?

A project is the main container for your entire Django web application.

- It holds global settings for your site (like database config, installed apps, middleware, etc.)
- It can contain multiple apps.
- Think of it as your overall website.

```
mysite/
│
├── mysite/        ← Project settings (main config)
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
│
├── manage.py      ← Project manager script
```

What is a Django App?

An app is a component or feature of the project.

- It performs a specific function (e.g., blog, users, products, orders).
- You can reuse the same app in other Django projects.
- Each app contains models, views, templates, and URLs related to one feature.

```
blog/
├── admin.py
├── apps.py
├── models.py     ← Database models
├── views.py      ← Business logic
├── urls.py       ← Routes
└── templates/    ← HTML files
```

What is LTS Support?

LTS stands for Long-Term Support.

In the context of Django (or any software):

- LTS versions receive security updates and critical bug fixes for an extended period (usually 3 years in Django).
- Non-LTS versions only get updates for a short time (~18 months).

What is pip freeze and its Use?

pip freeze is a command used to list all installed Python packages and their versions in your virtual environment.

```
pip freeze

<!-- Output Example: -->
Django==4.2
djangorestframework==3.14.0
pytz==2023.3
```

Create requirements.txt file:

```
pip freeze > requirements.txt
Reinstall all packages from file (on another system):

pip install -r requirements.txt
```

This is especially useful for sharing or deploying projects with the exact same dependencies.

Project structure

```
myproject/
│
├── manage.py
├── myproject/                  ← Project settings folder (same name as the
project)
│   ├── __init__.py
│   ├── settings.py             ← Global project config (database, apps,
middleware)
│   ├── urls.py                 ← Root URL routing
│   ├── asgi.py                 ← ASGI entry point (for async apps)
│   └── wsgi.py                 ← WSGI entry point (used by servers like
Gunicorn)
│
├── myapp/                      ← One functional app (e.g., watchlist)
│   ├── __init__.py             ← Treats folder as Python package
│   ├── admin.py                ← Registers models in Django admin
│   ├── apps.py                 ← App configuration class
│   ├── migrations/             ← Auto-generated DB schema files
│   │   └── __init__.py
│   ├── models.py               ← Database models (classes → tables)
│   ├── tests.py                ← Unit tests for the app
│   ├── views.py                  ← Logic for handling requests
│   ├── urls.py                   ← Routes URLs to views (you create this)
│
├── db.sqlite3                    ← Default lightweight database
└── requirements.txt              ← All installed packages (generated by `pip
freeze`)
```

python manage.py migrate

It applies database schema changes—like creating tables—for Django's built-in apps and your models.

Think of this as: "Make the database ready."

What it does:

- Creates the required tables in your database (like Excel sheets). It creates tables for:
- Your models (like Movie, Review)
- Django built-in apps (User, Admin, etc.)

When to use:

- First time after starting a project
- After adding new models or changing model fields

You wrote this model:

```
class Movie(models.Model):
    title = models.CharField(max_length=100)
```

Then you run:

```
python manage.py makemigrations
python manage.py migrate
```

This tells Django: "Create a table in the database for Movie."

python manage.py createsuperuser

It creates an admin user account to access the Django Admin interface. Django admin is a built-in tool that lets you manage your models via a web UI.

You need a superuser (admin) to log into /admin and manage data.

```
python manage.py createsuperuser
```

Then:

```
Username: admin
Email: admin@gmail.com
Password: ********
Password (again): ********
Superuser created successfully.
```

Now, you can visit: http://127.0.0.1:8000/admin and log in with admin and the password you gave.

What is models.py

models.py is where you define your database tables in Django. Each model is a Python class that represents a table in the database. Each attribute (field) in the model becomes a column in the table.

```
from django.db import models  # This line imports Django's model tools.

# Create your models here.
class Movie(models.Model):  # This creates a Movie model. Inherits from
models.Model, so Django knows this is a database model.
    name = models.CharField(max_length=50)
    description = models.CharField(max_length=200)
    active = models.BooleanField(default=True)

    def __str__(self):    #This makes the admin panel and shell show the movie
name instead of Movie object
        return self.name
```

The most commonly used Django model fields along with a short description. These are used in your models.py to define the structure of your database tables.

Basic Field Types

Field	Description
`CharField(max_length=...)`	For small to medium-length strings (e.g., name, title).
`TextField()`	For long text (e.g., description, content). No `max_length` required.
`IntegerField()`	Stores integers (whole numbers).
`FloatField()`	Stores floating-point numbers (e.g., 3.14).
`DecimalField(max_digits=..., decimal_places=...)`	For precise decimal values (e.g., money).

Boolean & Choices

Field	Description
`BooleanField()`	Stores `True` or `False`.
`NullBooleanField()`	Stores `True`, `False`, or `None` *(deprecated; use `BooleanField(null=True)`)*.
`ChoiceField()` *(not a real field)*	Use `choices` argument in fields like `CharField` or `IntegerField`.

Date & Time Fields

Field	Description
`DateField()`	Stores only a date (`YYYY-MM-DD`).
`TimeField()`	Stores only a time (`HH:MM`).
`DateTimeField()`	Stores date + time.
`DurationField()`	Stores a time duration (like `timedelta`).
`AutoField()`	Auto-incrementing primary key (used by default).
`BigAutoField()`	Like `AutoField`, but supports larger integers.

Files & Media

Field	Description
`FileField(upload_to='path/')`	For uploading files.
`ImageField(upload_to='path/')`	For uploading images (requires Pillow library).

Relationship Fields

Field	Description
`ForeignKey(Model, on_delete=...)`	Many-to-one relationship.
`OneToOneField(Model, on_delete=...)`	One-to-one relationship.
`ManyToManyField(Model)`	Many-to-many relationship.

Other Useful Fields

Field	Description
`EmailField()`	Stores and validates email addresses.
`URLField()`	Stores URLs.
`SlugField()`	For short labels typically used in URLs.
`UUIDField()`	Stores universally unique identifiers (UUID).

Field	Description
`GenericIPAddressField()`	Stores IPv4 or IPv6 addresses.
`JSONField()`	Stores structured JSON data (Django 3.1+).
`BinaryField()`	Stores binary data.
`PositiveIntegerField()`	Only allows positive integers.

what is admin.py

admin.py is a file inside your Django app folder where you register your app's models with Django's Admin site.

- The Admin site is a built-in web interface Django provides to manage your database data easily without writing any frontend code.
- By registering your models here, you allow them to be visible and manageable through the admin panel.

```
from django.contrib import admin
from .models import Movie

# Register your models here.
admin.site.register(Movie)
```

What is a View in Django

A view is a Python function or class that takes a web request and returns a web response. It contains the logic that determines what data gets displayed and how.

There are two types of views in Django:

- 1. Function-Based Views (FBV) – use plain functions
- 2. Class-Based Views (CBV) – use classes to encapsulate logic

Below example retuen JSON response of all elements. Means all the items/objects from movie object irrespective of number.

```
from django.shortcuts import render
from .models import Movie
from django.http import JsonResponse

# Create your views here. : Function based view
def movie_list(request):
    movies = Movie.objects.all()
    data = {'movies': list(movies.values())}
    print(movies.values())

    return JsonResponse(data)  # {"movies": [{"id": 1, "name": "Movie1",
"description": "Description-1", "active": true},
                               # {"id": 2, "name": "movie2", "description":
"Description-2", "active": false}]}
```

Now return specific element or individual element from the movie model

```
from django.shortcuts import render
from .models import Movie
from django.http import JsonResponse

def movie_detals(request, pk):
    movies = Movie.objects.get(pk=pk)
    data = {
        'name': movies.name,
        'description': movies.description,
        'active': movies.active
    }

    return JsonResponse(data) #{"name": "Movie1", "description": "Description-1",
"active": true}

app>urls.py

...
from django.urls import path, include
from .views import movie_list, movie_detals

urlpatterns = [
    path('list/', movie_list, name="movie-list"),
    path('<int:pk>', movie_detals, name="movie_detals"),
]
```

What is a QuerySet?

A QuerySet is a collection of objects from the database.

```
Movie.objects.all()
```

Example: Complex QuerySet

```
class Movie(models.Model):
    title = models.CharField(max_length=100)
    year = models.IntegerField()
    rating = models.FloatField()
    genre = models.CharField(max_length=50)
```

- Filter movies released after 2010 and genre is 'Action':

```
Movie.objects.filter(year__gt=2010, genre='Action')
```

- Order by rating (highest first):

```
Movie.objects.filter(genre='Action').order_by('-rating')
```

- Get only selected fields (like title and year):

```
Movie.objects.filter(genre='Comedy').values('title', 'year')
```

- Exclude some results:

```
Movie.objects.exclude(rating__lt=5.0)
```

- Combine filters using Q (OR condition):

```
from django.db.models import Q

Movie.objects.filter(Q(genre='Horror') | Q(rating__gte=8))
```

What is .values() in Django

.values() is a QuerySet method that returns dictionaries instead of full Django model objects.

```
<!-- Input : without value() -->
movies = Movie.objects.all()

<!-- Output -->
[<Movie: Inception>, <Movie: Interstellar>, ...]
```

```
<!-- with .value() -->
movies = Movie.objects.all().values()

<!-- output -->
```

```
[
  {'id': 1, 'title': 'Inception', 'year': 2010, 'rating': 8.8},
  {'id': 2, 'title': 'Interstellar', 'year': 2014, 'rating': 8.6},
]
```

.values() = "Give me plain data (dictionary), not full model objects."

urls.py in both the project and app

When someone visits your website, Django needs to know, “Where should I send this request?” This is where urls.py files come in.

Django Has Two Kinds of urls.py

- Project-level urls.py : Located in the main project folder (same as settings.py).
- App-level urls.py : You create this inside each app folder (optional but very useful).

(Project = House, App = Rooms)

```
myproject/
│
├── myproject/          ← Project folder
│   ├── urls.py         ← Project-level URL file
│   └── settings.py
│
├── movies/             ← App folder
│   └── urls.py         ← App-level URL file (you create this)
│   └── views.py
```

1. Project-level urls.py – the Main Gate This file connects the whole site to the right app.

```
# myproject/urls.py
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),              # Admin panel
    path('movies/', include('movies.urls')),      # Redirect /movies/ to the
movies app
]
```

If someone comes to /movies/, send them to the movies app’s url board.

imp : include('movies.urls'): content should be inside quotes.

2. App-level urls.py – the Room Directory This file connects the movie app's URLs to its views.

```
# movies/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('', views.movie_list),  # When user visits /movies/, show movie list
]
```

When user reaches the movies room, check what’s inside: Oh! Show the movie list.

```
Browser → Project urls.py → App urls.py → views.py → Output to browser
```