Django

---

1. Overview

Django is a high-level **Python web framework** that encourages rapid development and clean, pragmatic design.

- First released in 2005.
- Follows the **Model-Template-View (MTV)** architectural pattern (a variation of MVC).
- Batteries-included: comes with many built-in features like authentication, admin panel, ORM, and security.
- Promotes **DRY (Don't Repeat Yourself)** principle.
- Great for building scalable, secure, and maintainable web apps.

---

2. Architecture & Core Components

MTV Pattern

Component	Role
Model	Defines data structure and interacts with the database via Django ORM.
Template	Handles presentation layer (HTML, CSS, JS) with Django Template Language.
View	Contains business logic; processes requests and returns responses.

---

3. Django Project Structure

- **manage.py**: Command-line utility for managing the project.
- **settings.py**: Configuration file (DB, middleware, installed apps).
- **urls.py**: URL routing and dispatching.
- **wsgi.py / asgi.py**: Web server gateway interface for deployment.
- **apps/**: Modular components containing models, views, templates.

---

4. Models & ORM

- Define models as Python classes.
- Supports multiple databases (PostgreSQL, MySQL, SQLite, Oracle).
- Fields types: CharField, IntegerField, DateTimeField, ForeignKey, etc.
- Migration system tracks changes to schema and applies them.

Example:

```
from django.db import models


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

---

5. URL Routing

- URLs mapped to views via urls.py.
- Supports path converters, named URLs, and namespaces.

Example:

```
from django.urls import path
from . import views


urlpatterns = [
    path('', views.index, name='home'),
    path('post/<int:id>/', views.post_detail, name='post_detail'),
]
```

---

6. Views

- Handle request/response logic.
- Function-based views (FBVs) or Class-based views (CBVs).
- CBVs provide reusable generic views for common tasks (ListView, DetailView).

Example FBV:

```
from django.shortcuts import render


def index(request):
   return render(request, 'index.html')
```

---

7. Templates

- Use Django Template Language (DTL).
- Template inheritance (base templates and blocks).
- Template tags and filters for logic and formatting.

Example template:

```
{% extends "base.html" %}


{% block content %}
  <h1>{{ post.title }}</h1>
  <p>{{ post.body }}</p>
{% endblock %}
```

---

8. Forms

- Django forms handle validation, rendering, and data binding.
- ModelForms link forms directly to models.

Example:

```
from django import forms
from .models import Post


class PostForm(forms.ModelForm):
   class Meta:
      model = Post
```

```
    fields = ['title', 'body']
```

---

9. Authentication & Authorization

- Built-in user model with login, logout, password management.
- Permissions and groups for access control.
- Easily extendable for custom auth workflows.

---

10. Admin Interface

- Auto-generated admin dashboard for managing data.
- Customizable and extensible.
- Useful for content management and internal use.

---

11. Middleware

- Process requests/responses globally.
- Common uses: authentication, sessions, CSRF protection.

---

12. Deployment

- Commonly deployed with **WSGI** (Gunicorn, uWSGI).
- Supports **ASGI** for async applications.
- Integrates with Nginx or Apache as reverse proxy.
- Environment variables for secret management.
- Supports containerization (Docker).

---

13. Advantages

- Rapid development with batteries included.
- Strong community and ecosystem.
- Scalable and secure.
- Good documentation.

- ORM abstracts complex SQL queries.