DJANGO FOUNDATIONS – FULL EXPANDED DEFINITIONS & THEORY

1. Virtual Environment (venv)

A virtual environment (venv) is a completely isolated Python workspace created for a single project.

It has its own Python interpreter, its own pip, and its own site-packages folder. This prevents

different projects from interfering with each other.

Importance:

- Avoids dependency conflicts between projects.

- Ensures predictable deployments.

- Makes debugging easier because environment is isolated.

- Prevents global system pollution.

Internal Working:

- A folder named 'venv' is created with a dedicated Python binary.

- When activated, terminal switches to this Python.

- All packages install only inside venv, not system.

Final Definition:

A virtual environment is an isolated workspace allowing each project to maintain its own dependencies,

ensuring no conflict with global Python or other projects.

2. Django Project

A Django project is the master configuration of the entire web application. It contains global

settings, universal URLs, middleware configuration, database settings, static file rules,

and all apps that the system uses.

Key Components:

- manage.py: command-line tool for running server, migrations, etc.

- settings.py: stores database credentials, middleware, security settings.

- urls.py: root URL routing table.

- wsgi/asgi: server entry point for deployment.

Internal Flow:

Running 'python manage.py runserver' loads settings, apps, middleware, URL resolver, and starts

a lightweight server.

Final Definition:

A Django project is the complete configuration and structural foundation of a web application,

combining settings, URLs, middleware, and installed apps.

3. Django App

A Django app is a feature-specific module inside a project. It focuses on one task or functionality—

such as students, payments, authentication, blog, etc.

Why Apps Exist:

- They make the code modular.

- They allow reuse in other projects.

- They keep code organized and maintainable.

Structure Includes:

- models.py: database table definitions.

- views.py: logic and API handling.

- urls.py: app routes.

- admin.py: admin dashboard integration.

Final Definition:

A Django app is a reusable, independent module within a project dedicated to a single functionality.

4. HTTP Methods

HTTP methods define the client's intention when sending a request to the server.

Detailed Meanings:

- GET: Retrieve data without modifying anything.

- POST: Create new records in the database.

- PUT: Replace an entire resource.

- PATCH: Update only specific fields.

- DELETE: Remove existing resource.

Why Needed:

Without HTTP methods, server cannot differentiate between read/write/update/delete actions.

Final Definition:

HTTP methods represent standardized actions a client wants to perform on a server resource.

5. APIs in Django

An API (Application Programming Interface) is a backend function that accepts input,

processes logic, interacts with the DB, and returns structured output, usually JSON.

API Flow:

1. Client sends HTTP request.

2. Django view reads request body/query params.

3. Logic executes.

4. ORM interacts with DB.

5. JSON response is sent back.

Why APIs:

- Allows communication with React, Flutter, mobile apps.

- Makes backend independent of frontend.

- Enables modern full-stack development.

Final Definition:

An API is a communication layer that processes client requests, executes logic, and returns

JSON data to external systems.

6. Database Connection

Django connects to databases using DATABASES configuration in settings.py.

Supported Databases:

- SQLite (default)

- PostgreSQL (recommended for production)

- MySQL

- MariaDB

- Oracle

Migrations:

- makemigrations: detect model changes.

- migrate: apply schema changes.

Why Needed:

Every application requires persistent storage for data such as students, users, payments, logs, etc.

Final Definition:

Database connection is Django's configuration that allows the application to communicate with

a database via ORM and migrations.

7. ORM (Object Relational Mapping)

ORM allows developers to interact with the database using Python instead of SQL.

How It Works:

Python code:

Student.objects.create(name="Hari")

Internally becomes:

INSERT INTO student (name) VALUES ("Hari");

Why ORM:

- Prevents SQL injection.

- Portable across different databases.

- Cleaner and easier to maintain.

Final Definition:

ORM is Django's database abstraction layer that converts Python code into SQL queries automatically and securely.

8. Query Parameters

Query parameters are optional key-value pairs appended to URLs to provide additional data to the server.

Example:

/students?age=20&sort=name

Usage:

- Filtering

- Searching

- Pagination

- Sorting

Django Access:

request.GET.get("age")

Final Definition:

Query parameters are additional values attached to URLs that instruct the server to customize

its response based on filtering or search criteria.

9. MVT Architecture

MVT (Model-View-Template) is Django's core architecture.

Model:

- Represents database tables and fields.

- Handles data structure and relations.

View:

- Processes incoming requests.

- Calls database using ORM.

- Returns JSON or HTML response.

Template:

- HTML rendering engine (not used in APIs).

Flow:

User → URL → View → Model → View → Template/JSON → User

Final Definition:

MVT is Django's design pattern that separates data handling (Model), business logic (View),

and user interface (Template).

10. Middleware

Middleware is a system of hooks that process requests and responses globally before reaching

the view or returning to the client.

Use Cases:

- Logging

- Security checks

- Authentication

- Request modification

- Response customization

- IP blocking

- CORS

Execution Flow:

Incoming:

Client → Middleware 1 → Middleware 2 → View

Outgoing:

View → Middleware 2 → Middleware 1 → Client

Final Definition:

Middleware is Django's global request/response processing layer that allows custom logic to run automatically for every request.

11. Deployment

Deployment means hosting your Django application on a real server so that users can access it online.

Steps Include:

- Push code to GitHub.

- Configure cloud hosting (Render, AWS, Railway).

- Install dependencies.

- Set environment variables.

- Run migrations.

- Collect static files.

- Start production server (gunicorn/uvicorn).

- Configure domain + SSL.

Importance:

Production environments require more security, performance, stability, and monitoring

than development.

Final Definition:

Deployment is the process of preparing and running a Django project on a cloud server, making

it publicly accessible and production-ready.