**Meisum Abbas**

**TMRC**

**Task 3­­**

## What is a Virtual Environment?

A virtual environment is an isolated Python environment used for managing dependencies and packages per project.  
  
Example:  
You might need Django 4.2 for one project and Django 3.2 for another. If you install everything globally, one project may break due to version mismatch.  
  
Using a virtual environment keeps dependencies separated.  
  
Commands:  
 python -m venv env # Create virtual environment  
 env\Scripts\activate # Activate on Windows  
 source env/bin/activate # Activate on Mac/Linux  
 deactivate # Exit virtual environment

## What Problem Does a Virtual Environment Solve?

It solves the problem of dependency conflicts by isolating each project's packages.  
  
Example:  
Project A needs NumPy 1.19 and Project B needs NumPy 1.25. A virtual environment ensures both projects can work independently without clashing.

## How Do We Use a Virtual Environment?

Steps to use:  
1. Create:  
 python -m venv env  
2. Activate:  
 env\Scripts\activate (Windows)  
 source env/bin/activate (Linux/Mac)  
3. Install packages inside:  
 pip install django  
4. Deactivate:  
 deactivate

## What is Django?

Django is a high-level Python web framework that simplifies building secure and maintainable websites quickly.  
  
Features:  
- Admin Panel  
- ORM (Object Relational Mapping)  
- Authentication System  
- URL Routing  
  
Example:  
To build a blog, Django provides built-in tools to create posts, manage users, and render pages without coding everything from scratch.

## Difference Between Django and Django REST Framework (DRF)

Django is used for full web applications (with HTML), while Django REST Framework is used to build APIs (returns JSON).  
  
Example:  
- Django: Show product page with HTML + CSS  
- DRF: Provide product data as JSON for a mobile app  
  
Tools in DRF include:  
- Serializers  
- ViewSets  
- Routers

## Django Architecture

Django follows MTV (Model-Template-View) architecture:  
  
- Model: Python classes that map to database tables.  
- Template: HTML files that show data.  
- View: Python functions or classes that contain logic.  
  
Example:  
View gets a product from Model → sends it to Template → Template shows HTML to user.

## Django Code-Level Flow

Flow:  
1. User requests a URL (e.g., /products/)  
2. Django matches the URL in urls.py  
3. Related view function runs from views.py  
4. View accesses data using models.py  
5. Data passed to template (HTML)  
6. Template rendered and sent to browser  
  
Example Code:  
urls.py → maps '/products/' → views.product\_list → calls Product.objects.all() → renders product\_list.html

## Django Request-Response Cycle

Cycle:  
1. User sends HTTP request  
2. URLConf matches URL  
3. View is triggered  
4. View may fetch from DB via model  
5. Data passed to HTML template  
6. Rendered page sent back as HTTP response  
  
Example:  
Request: /blog/  
→ Match URL  
→ Call blog\_list() in views.py  
→ Fetch posts  
→ Render blog\_list.html  
→ Show to user

## Difference Between Django App and Django Project

- Django Project: Entire website setup (settings, configurations, multiple apps)  
- Django App: A modular feature (like 'blog', 'users', 'cart')  
  
Example:  
- Project: E-commerce site  
- Apps: Products, Cart, Checkout, User  
  
Commands:  
 django-admin startproject mysite  
 python manage.py startapp blog

## Understanding Django App and Project File Structure

Project Structure (after startproject):  
mysite/  
├── manage.py  
├── mysite/  
│ ├── settings.py  
│ ├── urls.py  
  
App Structure (after startapp blog):  
blog/  
├── models.py  
├── views.py  
├── urls.py  
├── templates/  
├── admin.py  
  
- settings.py: Main config file  
- urls.py: Routing  
- models.py: Database tables  
- views.py: Logic  
- templates/: HTML files

## Explore Django Admin Panel

Django provides a built-in admin panel to manage database records via UI.  
  
Steps to Use:  
1. Add your app to INSTALLED\_APPS in settings.py  
2. Run migrations:  
 python manage.py migrate  
3. Create superuser:  
 python manage.py createsuperuser  
4. Register models in admin.py:  
 from .models import Product  
 admin.site.register(Product)  
5. Access via: http://127.0.0.1:8000/admin  
  
Example:  
Once Product model is registered, you can add/edit/delete products via admin panel.