Introduction of Django and Its Installation

Class 20

12/7/2025

Acknowledgement

The series of the IT & Japanese language course is Supported by AOTS and OEC.

Ministry of Economy, Trade and Industry

Overseas Employment Corporation

What you have Learnt Last Week

We were focused on following points.

- Usage of control and loop flow statement
- Performing Linear Algebra in Numpy
- Why Requirement Analysis is so important in the process?
- Software development Life cycle
- Importance of Security compliance
- Introduction of Bash Scripting
- Introduction of docker and docker compose
- Introduction of Ansible

What you will Learn Today

We will focus on following points.

1. URL Routing And Django Apps
2. Get vs Post METHOD In Django
3. Django Admin Panel & Manipulation Of Database
4. Run and Deploy A Blog With Django
5. Quiz
6. Q&A Session

Django Project vs App Structure

Understand the Basic Structure

In Django, a **Project** is the entire web application. An **App** is a part or module inside that project. You can think of a project like a full website and apps as features such as blog, login system, comments, etc.

Commands:

```
git clone https://github.com/tomitokko/django-blog.git
django-admin startproject mysite
cd mysite
python manage.py startapp blog
```

Folder Structure Example:

```
mysite/          # Project
  ├──── blog/      # App
  ├──── manage.py
  └──── mysite/   # Project settings folder
```

Registering a Django App

Make Django Recognize Your App

Step-by-Step:

1. Go to mysite/settings.py
2. Find the INSTALLED_APPS list
3. Add your app name — in this case, 'blog

Example:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    ...
    'blog',  # ✅ Register your app here
]
```

Creating Views in the App

Add Basic View Logic

Go to blog/views.py and write a function-based view.

Example:

```
from django.http import HttpResponse


def home(request):
    return HttpResponse("Hello from Blog App!")
```

Creating URLs for the App

Setup URL Routing for Your App

- Inside your blog/ folder, create a file: urls.py
- Define the route for the view

Example (blog/urls.py):

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

urlpatterns = [
    path('', views.home, name='home'),
]
```

Connect App URLs to Project

Link App Routing to Project Routing

Open mysite/urls.py and include the blog’s routing file.

Example (mysite/urls.py):

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('blog/', include('blog.urls')),  # ✅ Add
this line
]
```

Now if you visit http://localhost:8000/blog/ → it shows “Hello from Blog App!”

Using Dynamic URL Path Converters

Create URLs with Variables

You can use special tags to make your URL dynamic.

Example:

```
path('post/<int:id>/', views.post_detail, name='post-detail')
```

In views.py, define:

```
def post_detail(request, id):
    return HttpResponse(f"Viewing post with
ID: {id}")
```

🌐 **Visiting /blog/post/10/ will show:**

```
Viewing post with ID: 10
```

Understanding GET vs POST

Two Common HTTP Methods in Django Forms

- **GET**: Used to retrieve data from the server
- **POST**: Used to send data to the server for processing
- Forms can use either method depending on the use case

Example:

```
<form method="GET"> or <form method="POST">
```

Creating a Form in Django

Basic HTML Form Structure

```
<form method="POST">
  {% csrf_token %}
  <input type="text" name="username">
  <button type="submit">Submit</button>
</form>
```

- Always include {% csrf_token %} for security
- Action URL can point to your view (or left empty for same page)

Handling GET Requests in Views

Use request.GET to Fetch Query Parameters

```
def search_view(request):
    if request.method == "GET":
        query = request.GET.get('search_term')
        return render(request, 'results.html', {'query': query})
```

- Data is visible in the browser URL
- Used for search boxes, filters, etc.

Handling POST Requests in Views

Use request.POST to Handle Form Submission

```
def submit_form(request):
    if request.method == "POST":
        username = 
request.POST.get('username')
        return render(request, 'thanks.html', 
{'username': username})
```

- Data is **not visible** in URL
- Used for login forms, registration, etc.

CSRF Token and Security

Why {% csrf_token %} is Important

- **CSRF (Cross-Site Request Forgery)** protection
- Required for POST forms
- Prevents external sites from submitting forms on your behalf

Always add in your form:

```
{% csrf_token %}
```

Summary and Use Cases

When to Use GET vs POST

Method	Use For	Data in URL	Secure?
GET	Search, filters	Yes	Less secure
POST	Login, Register	No	More secure

Introduction to Django Admin Panel

Manage Your App with a Web Interface

- Django provides a built-in admin panel
- Automatically generated after migrations
- Allows CRUD operations (Create, Read, Update, Delete)

Access:

Run server and go to http://127.0.0.1:8000/admin

Login using superuser credentials

Registering Models in admin.py

Show Models in the Admin Panel

```
# blog/admin.py
from django.contrib import admin
from .models import Post


admin.site.register(Post)
```

- Add each model you want to manage
- Appears in the admin dashboard

Customizing Admin Panel Display

Improve How Data Appears

```
@admin.register(Post)

class PostAdmin(admin.ModelAdmin):

    list_display = ('title', 'author', 'created_at')

    search_fields = ('title', 'content')
```

- list_display: Show these fields in table
- search_fields: Adds a search bar

Performing CRUD in Admin Panel

Create, Update, Delete Made Easy

- Click **Add** to create a new record
- Click on an item to edit
- Check and use **delete selected** to remove

No code required – all from web interface!

Working with Django Shell

Use Python Shell to Interact with DB

```
python manage.py shell
```

```
from blog.models import Post
Post.objects.all()
Post.objects.create(title="Hello", content="World")
```

- Great for testing without UI
- Supports full query power

Introduction to Blog Deployment

From Local to Live

Project: https://github.com/tomitokko/django-blog

Goal: Run the blog locally, then deploy it on AWS EC2

Tech Stack: Django, Python, EC2 (Ubuntu)

Clone and Setup Project Locally

Run Django Blog on Local Machine

Step1: Clone the repo:

```
git clone https://github.com/tomitokko/django-blog.git
cd django-blog
```

Step2: Create virtual environment:

```
python3 -m venv env

source env/bin/activate
```

Step3: Install dependencies:

```
pip install -r requirements.txt
```

Clone and Setup Project Locally

Run Django Blog on Local Machine

Step4: Run migrations:

```
python manage.py makemigrations

python manage.py migrate
```

Step5: Run aServer:

```
python manage.py runserver
```

Step6: Access on: http://127.0.0.1:8000

Use **python manage.py createsuperuser**
to access the admin panel Ensure all routes and pages work

Set Up AWS EC2 Instance

Configure EC2 for Django App

Step1: Launch EC2 Ubuntu instance

Step2: into instance:

```
ssh -i key.pem ubuntu@your-ec2-public-ip
```

Step3: Update packages:

```
sudo apt update
```

Step4: Install Python, pip, virtualenv, Git:

```
sudo apt install python3-pip python3-venv git -y
```

Deploy Project to EC2 Subtitle

Run on EC2

Step 5: Clone the repo on EC2

```
git clone https://github.com/tomitokko/django-blog.git
cd django-blog
```

Step 6: Set up virtualenv and install

```
python3 -m venv env
source env/bin/activate
pip install -r requirements.txt
```

Step 7: Run migrations & start server:

```
python manage.py migrate
python manage.py runserver 0.0.0.0:8000
```

Note: Add EC2 public IP to ALLOWED_HOSTS in settings.py

Add Security Group Rule and Test

Allow Global Access

- In AWS Console > EC2 > Security Groups
- Add inbound rule:
    - Type: Custom TCP
    - Port: 8000
    - Source: 0.0.0.0/0
- Visit: http://your-ec2-public-ip:8000
- App should be live!

Create Views

Display and Create Blog Posts

```
# blog/views.py
from .models import Post
from django.shortcuts import render

def post_list(request):
    posts = Post.objects.all()
    return render(request, 'post_list.html', {'posts': posts})

def post_detail(request, id):
    post = Post.objects.get(id=id)
    return render(request, 'post_detail.html', {'post': post})
```

Optional: Add a form to create posts

Run on Localhost

Test Before Deployment

```
python manage.py runserver
```

- Visit http://127.0.0.1:8000/
- Check all views and templates
- Test data creation via Admin Panel or custom form

Final Checklist Before Deployment

Common Tasks

✅ ALLOWED_HOSTS in settings.py

✅ Install gunicorn for production

✅ Add static file support (collectstatic)

✅ Secure secrets using .env files

✅ Debug OFF for production

Assignment

Assignment

Commands

Upload this whole project into your GitHub profile

with proper documentation

Quiz Section

Quiz

Everyone student should click on submit button before time ends otherwise MCQs will not be submitted

[Guidelines of MCQs]

1. There are 20 MCQs
2. Time duration will be 10 minutes
3. This link will be share on 12:25pm (Pakistan time)
4. MCQs will start from 12:30pm (Pakistan time)
5. This is exact time and this will not change
6. Everyone student should click on submit button otherwise MCQs will not be submitted after time will finish
7. Every student should submit Github profile and LinkedIn post link for every class. It include in your performance

Assignment

Assignment should be submit before the next class

[Assignments Requirements]

1. Create a post of today's lecture and post on LinkedIn.
2. Make sure to tag @Plus W @Pak-Japan Centre and instructors LinkedIn profile
3. Upload your code of assignment and lecture on GitHub and share your GitHub profile in respective

   your region group WhatsApp group
4. If you have any query regarding assignment, please share on your region WhatsApp group.
5. Students who already done assignment, please support other students

Q&A Session

ありがとうございます。

Thank you.

شکریا

+W

For the World with Diverse Individualities