Let’s turn this into a **clear, repeatable daily Django warm-up with real code + explanations**. This is designed so a beginner can **follow the same flow every day and actually understand what’s happening**.

□ Step 0: Create Project (only once)

Run this once if you don’t already have a project:

```
django-admin startproject myproject
cd myproject
python manage.py runserver
```

□ Open browser: `http://127.0.0.1:8000/`
If it works → you're ready

□ DAILY WARM-UP (Follow This Same Flow Every Day)

□ Step 1: Create App

```
python manage.py startapp posts
```

□ This creates a folder:

```
posts/
    models.py
    views.py
    admin.py
```

□ Why?

Django uses **apps** to organize features.
Think: one app = one feature (blog, users, payments)

□ Step 2: Register App in Settings

Open `settings.py`

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'posts',  # □ add this
]
```

□ Why?

Django won't recognize your app unless you register it here.

□ Step 3: Create a Model (Database Table)

Open `posts/models.py`

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()

    def __str__(self):
        return self.title
```

□ Explanation:

- `Post` → table in database
- `title` → short text field
- `content` → long text
- `__str__` → how it shows in admin panel

□ Step 4: Migrations (VERY IMPORTANT)

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

□ Why?

- `makemigrations` → creates instructions

- `migrate` → applies them to database

 Without this, your model DOES NOT exist in DB

Step 5: Show Model in Admin Panel

Open `posts/admin.py`

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

admin.site.register(Post)
```

Now create admin user:

```
python manage.py createsuperuser
```

Run server and go to:

```
http://127.0.0.1:8000/admin/
```

 Login → You'll see **Post table**

Step 6: Create a View (Logic)

Open `posts/views.py`

```
from django.http import HttpResponse

def home(request):
    return HttpResponse("Hello, this is my first Django page!")
```

What is a view?

A **view = function that handles request and returns response**

Step 7: Connect URL to View

Create `posts/urls.py`

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

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

Connect to main project (`myproject/urls.py`)

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('posts.urls')),
]
```

□ Flow:

Browser → URL → View → Response

□ Step 8: Use Templates (HTML)

Create folder structure:

```
posts/
    templates/
        home.html
```

`home.html`

```
<h1>{{ message }}</h1>
```

Update `views.py`

```
from django.shortcuts import render

def home(request):
    return render(request, 'home.html', {
        'message': 'Hello from Django Template!'
    })
```

□ Why templates?

Instead of hardcoding HTML in Python, we use separate HTML files.

□ Step 9: Show Data from Database

Update `views.py`:

```
from .models import Post

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

Update `home.html`

```
<h1>All Posts</h1>

{% for post in posts %}
    <h2>{{ post.title }}</h2>
    <p>{{ post.content }}</p>
{% endfor %}
```

□ What's happening?

- Fetch data from DB
- Send it to template
- Loop and display

Here's a **complete Django warm-up practice project (FULL UPDATED CRUD + LOGIN/LOGOUT using ModelForm)** in a clean step-by-step flow.

This is designed so you can **repeat it daily until it becomes automatic memory**.

□ "Mini Blog App" (Production-style Django basics)

Includes:

- □ Signup / Login / Logout
- □ Create / Read / Update / Delete posts
- □ ModelForm (clean Django way)
- □ Logged-in user ownership

□ STEP 1 — CREATE PROJECT

```
django-admin startproject myproject
cd myproject
python manage.py startapp posts
python manage.py startapp accounts
```

□ STEP 2 — SETTINGS

myproject/settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'posts',
    'accounts',
]
```

□ □ STEP 3 — MODEL (DATABASE)

posts/models.py

```
from django.db import models
from django.contrib.auth.models import User

class Post(models.Model):
    title = models.CharField(max_length=120)
    content = models.TextField()
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title
```

□ STEP 4 — MIGRATIONS

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

□ STEP 5 — MODEL FORM (IMPORTANT)

posts/forms.py

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

class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ['title', 'content']
```

□ STEP 6 — AUTH (SIGNUP SYSTEM)

accounts/views.py

```
from django.contrib.auth.forms import UserCreationForm
from django.shortcuts import render, redirect

def signup(request):
    if request.method == "POST":
        form = UserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('login')
    else:
        form = UserCreationForm()

    return render(request, 'signup.html', {'form': form})
```

---

accounts/urls.py

```
from django.urls import path
from .views import signup
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('signup/', signup, name='signup'),
    path('login/', auth_views.LoginView.as_view(template_name='login.html'),
name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
]
```

□ STEP 7 — MAIN URLS

myproject/urls.py

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('accounts.urls')),
    path('', include('posts.urls')),
]
```

□ STEP 8 — CRUD VIEWS (MODELFORM WAY)

posts/views.py

```
from django.shortcuts import render, redirect, get_object_or_404
from .models import Post
from .forms import PostForm
from django.contrib.auth.decorators import login_required

# READ
def post_list(request):
    posts = Post.objects.all().order_by('-created_at')
    return render(request, 'post_list.html', {'posts': posts})


# CREATE
@login_required
def create_post(request):
    if request.method == "POST":
        form = PostForm(request.POST)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user
            post.save()
            return redirect('post_list')
    else:
        form = PostForm()

    return render(request, 'create_post.html', {'form': form})


# UPDATE
@login_required
def update_post(request, id):
    post = get_object_or_404(Post, id=id)

    if request.method == "POST":
        form = PostForm(request.POST, instance=post)
        if form.is_valid():
            form.save()
            return redirect('post_list')
    else:
        form = PostForm(instance=post)

    return render(request, 'update_post.html', {'form': form})


# DELETE
@login_required
def delete_post(request, id):
    post = get_object_or_404(Post, id=id)
```

```
    if request.method == "POST":
        post.delete()
        return redirect('post_list')

    return render(request, 'delete_post.html', {'post': post})
```

□ STEP 9 — POSTS URLS

posts/urls.py

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

urlpatterns = [
    path('', views.post_list, name='post_list'),
    path('create/', views.create_post, name='create_post'),
    path('update/<int:id>/', views.update_post, name='update_post'),
    path('delete/<int:id>/', views.delete_post, name='delete_post'),
]
```

□ STEP 10 — TEMPLATES

Create folder:

```
posts/templates/
accounts/templates/
```

□ posts/templates/post_list.html

```
<h1>Blog Posts</h1>

<a href="{% url 'create_post' %}">Create Post</a>

{% for post in posts %}
    <h2>{{ post.title }}</h2>
    <p>{{ post.content }}</p>
    <small>By {{ post.author }}</small>

    <a href="{% url 'update_post' post.id %}">Edit</a>
    <a href="{% url 'delete_post' post.id %}">Delete</a>
    <hr>
{% endfor %}
```

□ posts/templates/create_post.html

```
<h2>Create Post</h2>

<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Save</button>
</form>
```

☐ posts/templates/update_post.html

```
<h2>Update Post</h2>

<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Update</button>
</form>
```

☐ posts/templates/delete_post.html

```
<h2>Delete Post</h2>

<p>Are you sure you want to delete "{{ post.title }}"?</p>

<form method="post">
    {% csrf_token %}
    <button type="submit">Yes Delete</button>
</form>
```

☐ accounts/templates/signup.html

```
<h2>Signup</h2>

<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Signup</button>
</form>
```

---

accounts/templates/login.html

```
<h2>Login</h2>

<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Login</button>
</form>
```

STEP 11 — LOGIN SETTINGS

settings.py

```
LOGIN_REDIRECT_URL = '/'
LOGOUT_REDIRECT_URL = '/accounts/login/'
```

▶ STEP 12 — RUN PROJECT

```
python manage.py runserver
```

Open:

- /accounts/signup/
- /accounts/login/
- /create/
- /

Right now, **any logged-in user can edit/delete any post**.

We'll fix that so:

 A user can ONLY update or delete THEIR OWN posts

GOAL

- Author creates post → saved with `request.user`
- Only that same user can edit/delete it
- Others get blocked (403 Forbidden or redirect)

STEP 1 — MODEL (already correct)

Make sure your model has `author`:

```
from django.db import models
from django.contrib.auth.models import User

class Post(models.Model):
    title = models.CharField(max_length=120)
    content = models.TextField()
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
```

□ STEP 2 — CREATE POST (OWNER ASSIGNMENT)

views.py

```
@login_required
def create_post(request):
    if request.method == "POST":
        form = PostForm(request.POST)
        if form.is_valid():
            post = form.save(commit=False)
            post.author = request.user  # □ IMPORTANT: ownership assigned
here
            post.save()
            return redirect('post_list')
    else:
        form = PostForm()

    return render(request, 'create_post.html', {'form': form})
```

□ STEP 3 — SECURITY RULE (MOST IMPORTANT)

We will restrict **UPDATE + DELETE**

Option A (BEST PRACTICE): 403 Forbidden

views.py

UPDATE (User-specific, secure)

```
from django.shortcuts import get_object_or_404, render, redirect
from django.contrib.auth.decorators import login_required
from .models import Post
from .forms import PostForm

@login_required
def post_update(request, pk):
    # □ only fetch post owned by logged-in user
    post = get_object_or_404(Post, pk=pk, author=request.user)

    if request.method == "POST":
        form = PostForm(request.POST, instance=post)
        if form.is_valid():
            form.save()
            return redirect('post_list')
    else:
        form = PostForm(instance=post)
```

```
    return render(request, 'posts/post_form.html', {'form': form})
```

□ What this does

- `get_object_or_404(Post, pk=pk, author=request.user)`
  → ensures user can ONLY access their own post
- No extra `if post.author != request.user` needed
  □ because query already blocks access

□ DELETE (User-specific, secure)

```
from django.shortcuts import get_object_or_404, redirect
from django.contrib.auth.decorators import login_required
from .models import Post

@login_required
def post_delete(request, pk):
    # □ only allow owner to fetch object
    post = get_object_or_404(Post, pk=pk, author=request.user)

    if request.method == "POST":
        post.delete()
        return redirect('post_list')

    return render(request, 'posts/post_confirm_delete.html', {'post': post})
```

□ STEP 4 — HIDE EDIT/DELETE BUTTONS (FRONTEND SAFETY)

Even though backend is secure, also hide UI:

post_list.html

```
{% for post in posts %}
    <h2>{{ post.title }}</h2>
    <p>{{ post.content }}</p>

    {% if post.author == request.user %}
        <a href="{% url 'update_post' post.id %}">Edit</a>
        <a href="{% url 'delete_post' post.id %}">Delete</a>
    {% endif %}

{% endfor %}
```

□ WHY BOTH BACKEND + FRONTEND?

Layer	Purpose
Frontend check	UX (hide buttons)
Backend check	Security (real protection)

□ NEVER rely only on HTML conditions.

□ RESULT YOU NOW HAVE

✔ User creates own posts
✔ User sees only own edit/delete buttons
✔ Backend blocks unauthorized access
✔ Secure CRUD system (real Django pattern)