# Django 5 – TemplateView (Generic Class-Based View)

## 1. What is TemplateView?

* TemplateView is a **Generic Class-Based View** (GCBV) used to render a **template**.
* It saves us from writing boilerplate code (render() every time).
* Works best for **static pages** (home, about, contact) or **pages with simple context data**.

👉 Import:

from django.views.generic.base import TemplateView

## 2. Different Ways to Use TemplateView

### 🔹 Method 1: Directly in urls.py

👉 Use TemplateView.as\_view() directly without creating a view in views.py.

# urls.py

from django.urls import path

from django.views.generic.base import TemplateView

urlpatterns = [

    path('home/', TemplateView.as\_view(template\_name='myapp/home.html'), name='home'),

]

✅ Best for **static pages** where no extra logic is required.

### 🔹 Method 2: Import from Views (Still using TemplateView.as\_view())

# views.py

from django.views.generic.base import TemplateView

# urls.py

from django.urls import path

from myapp import views

urlpatterns = [

    path('index/', views.TemplateView.as\_view(template\_name='myapp/index.html'), name='index'),

]

⚠️ Here, we are still **not creating a custom class**.  
We’re just using TemplateView directly.

### 🔹 Method 3: Subclassing TemplateView

👉 Create a **custom view class** that inherits from TemplateView.

# views.py

from django.views.generic.base import TemplateView

class AboutTemplateView(TemplateView):

    template\_name = 'myapp/about.html'

# urls.py

from django.urls import path

from myapp import views

urlpatterns = [

    path('about/', views.AboutTemplateView.as\_view(), name='about'),

]

✅ This is the **preferred way** when you want to **add extra logic** later.

## 3. Adding Context Data in TemplateView

### 🔹 Method A: Override get\_context\_data()

# views.py

class ContactTemplateView(TemplateView):

    template\_name = 'myapp/contact.html'

    def get\_context\_data(self, \*\*kwargs):

        context = super().get\_context\_data(\*\*kwargs)

        context["name"] = "Sonam"

        context["roll"] = 101

        return context

# urls.py

path('contact/', views.ContactTemplateView.as\_view(), name='contact'),

👉 In contact.html, you can use:

<p>Name: {{ name }}</p>

<p>Roll: {{ roll }}</p>

### 🔹 Method B: Passing Extra Context through urls.py

# urls.py

path(

    'contact/',

    views.ContactTemplateView.as\_view(extra\_context={'course': 'Python'}),

    name='contact'

)

👉 In contact.html:

<p>Course: {{ course }}</p>

✅ Best for small context that doesn’t require logic.

## 4. Passing Dynamic Data (with URL Parameters)

👉 Example: A profile page with a dynamic id.

# views.py

class ProfileTemplateView(TemplateView):

    template\_name = 'myapp/profile.html'

    def get\_context\_data(self, \*\*kwargs):

        context = super().get\_context\_data(\*\*kwargs)

        context["name"] = "TALHA"

        context["id"] = self.kwargs['id']   # 🔹 dynamic from URL

        print(context)

        return context

# urls.py

path('profile/<int:id>/', views.ProfileTemplateView.as\_view(), name='profile'),

👉 In profile.html:

<p>Name: {{ name }}</p>

<p>ID: {{ id }}</p>

✅ This way, you can access **URL parameters** inside your templates.

## 5. Summary – When to Use Which Method?

| **Method** | **Usage** |
| --- | --- |
| **Method 1 (Direct in URL)** | For very simple static pages (no extra logic). |
| **Method 2 (Import TemplateView)** | Same as Method 1, but imported in views.py. |
| **Method 3 (Subclassing)** | Best practice when you need extra logic or want to extend later. |
| **get\_context\_data()** | Use when you need to add dynamic or extra context. |
| **extra\_context in urls.py** | Quick way to pass small static context. |

## 6. Advantages of TemplateView

✅ Less code (no need to manually use render).  
✅ Clean separation of logic (context in get\_context\_data()).  
✅ Reusable across multiple templates.  
✅ Can handle static + dynamic content.