**Video file : 90**

**Code file : ch80**

**Generic Class based view ListView in Django 5**

# Notes on ****Generic Class-Based View: ListView in Django 5****

## ✅ 1. What is ListView?

* ListView is a **Generic Class-Based View (GCBV)** in Django that displays a list of objects from a model.
* Instead of writing repetitive queries and passing data to templates manually, Django provides this **ready-made view**.
* By default:
  + **Template name** → <app>/<model>\_list.html
  + **Context variable** → object\_list (also <model>\_list if a model is given)

## ✅ 2. Inheritance Chain of ListView

The power of ListView comes from multiple inheritance:

ListView → BaseListView → MultipleObjectMixin → View

→ TemplateResponseMixin

* **View** → The most basic class that handles HTTP requests (get(), post(), etc.).
* **MultipleObjectMixin** → Provides features for handling multiple objects (like .get\_queryset(), pagination, ordering).
* **TemplateResponseMixin** → Provides rendering logic using templates (template\_name, get\_template\_names()).
* **BaseListView** → Combines View + MultipleObjectMixin.
* **ListView** → Final class that combines BaseListView + TemplateResponseMixin.

👉 So, ListView can filter, paginate, order objects, and render them via a template automatically.

## ✅ 3. Comparison: Without vs With ListView

### Function-Based View (Manual Work)

# views.py

from django.shortcuts import render

from django.views import View

from myapp.models import Student

class AllStudentView(View):

    def get(self, request):

        all\_students = Student.objects.all()

        return render(request, 'myapp/all\_student.html', {'all\_students': all\_students})

# urls.py

path('', AllStudentView.as\_view(), name='all\_student')

### Using ListView (Much Easier)

# views.py

from django.views.generic.list import ListView

from myapp.models import Student

class StudentListView(ListView):

    model = Student

# urls.py

path('students/', StudentListView.as\_view(), name='students')

* Default template expected: student\_list.html
* Default context: object\_list and student\_list

## ✅ 4. Example Template

<!-- student\_list.html -->

<!DOCTYPE html>

<html>

<head><title>Students</title></head>

<body>

  <!-- Using student\_list -->

  {% for stu in student\_list %}

    <h3>{{ stu.name }} | {{ stu.roll }} | {{ stu.course }}</h3>

  {% endfor %}

  <hr>

  <!-- Using object\_list -->

  {% for stu in object\_list %}

    <h3>{{ stu.name }} | {{ stu.roll }} | {{ stu.course }}</h3>

  {% endfor %}

</body>

</html>

## ✅ 5. Customizing Template Names

### (a) Using template\_name\_suffix

class StudentListView1(ListView):

    model = Student

    template\_name\_suffix = '\_all'   # Default would be student\_list.html → becomes student\_all.html

Template: student\_all.html

### (b) Using template\_name

class StudentListView2(ListView):

    model = Student

    template\_name = 'myapp/students.html'   # Fully custom template name

## ✅ 6. Changing Context Variable

class StudentListView3(ListView):

    model = Student

    template\_name = 'myapp/students.html'

    context\_object\_name = 'students'   # Instead of 'object\_list' or 'student\_list'

<!-- students.html -->

{% for stu in students %}

  <h3>{{ stu.name }} | {{ stu.roll }} | {{ stu.course }}</h3>

{% endfor %}

## ✅ 7. Ordering and Filtering with get\_queryset()

class StudentListView4(ListView):

    model = Student

    template\_name = 'myapp/sabhistudents.html'

    context\_object\_name = 'students'

    ordering = ['name']   # Order by name

    def get\_queryset(self):

        return Student.objects.filter(course='Python')  # Filtered queryset

## ✅ 8. Adding Extra Context with get\_context\_data()

class StudentListView5(ListView):

    model = Student

    template\_name = 'myapp/sabhistudents.html'

    context\_object\_name = 'students'

    def get\_queryset(self):

        return Student.objects.filter(course='Python')

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

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

        context["django\_students"] = Student.objects.filter(course='Django')

        return context

In template:

{% for stu in django\_students %}

  <p>{{ stu.name }} | {{ stu.course }}</p>

{% endfor %}

## ✅ 9. Changing Template Dynamically with get\_template\_names()

class StudentListView6(ListView):

    model = Student

    template\_name = 'myapp/default.html'

    def get\_template\_names(self):

        if self.request.COOKIES.get('user') == 'sonam':

            return ['myapp/sonam.html']

        return [self.template\_name]

# 🎯 Final Summary for Notes

* **ListView** is a Generic Class-Based View for listing objects.
* **Inheritance**: ListView → BaseListView → (View + MultipleObjectMixin) + TemplateResponseMixin.
* **Defaults**:
  + Template → <app>/<model>\_list.html
  + Context → object\_list & <model>\_list
* **Customization options**:
  + template\_name\_suffix → change only suffix (\_all, \_details).
  + template\_name → give full custom template.
  + context\_object\_name → change context variable.
  + ordering → default ordering.
  + get\_queryset() → filter/modify data.
  + get\_context\_data() → add extra context.
  + get\_template\_names() → dynamically choose template.
* Makes code **cleaner, shorter, reusable** compared to FBVs.