**DetailView – Class Based Views Django**

Detail View refers to a view (logic) to display one instances of a table in the database. We have already discussed basics of Detail View in [Detail View – Function based Views Django](https://www.geeksforgeeks.org/detail-view-function-based-views-django/). Class-based views provide an alternative way to implement views as Python objects instead of functions. They do not replace function-based views, but have certain differences and advantages when compared to function-based views:

* Organization of code related to specific HTTP methods (GET, POST, etc.) can be addressed by separate methods instead of conditional branching.
* Object oriented techniques such as mixins (multiple inheritance) can be used to factor code into reusable components.
* Class based views are simpler and efficient to manage than function-based views. A function based view with tons of lines of code can be converted into a class based views with few lines only. This is where Object Oriented Programming comes into impact.
* **Django DetailView – Class Based Views**
* Illustration of **How to create and use Detail view** using an Example. Consider a project named scs having an app named scsdbapp.

Models.py

# import the standard Django Model

# from built-in library

from django.db import models

# declare a new model with a name "SCSModel"

class SCSModel(models.Model):

# fields of the model

title = models.CharField(max\_length = 200)

description = models.TextField()

# renames the instances of the model

# with their title name

def \_\_str\_\_(self):

return self.title

Python manage.py [makemigrations](https://www.geeksforgeeks.org/django-app-model-python-manage-py-makemigrations-command/)

Python manage.py [migrate](https://www.geeksforgeeks.org/django-manage-py-migrate-command-python/)

Python manage.py shell

from scsdbapp.models import SCSModel

>>> SCSModel.objects.create(

title="title1",

description="description1").save()

>>> SCSModel.objects.create(

title="title2",

description="description2").save()

>>> SCSModel.objects.create(

title="title2",

description="description2").save()

create views.py

Class Based Views automatically setup everything from A to Z. One just needs to specify which model to create DetailView for, then Class based DetailView will automatically try to find a template in app\_name/modelname\_detail.html. In our case it is geeks/templates/scsdbapp/SCSModel\_detail.html. Let’s create our class based view. In scsdbapp/views.py,

from django.views.generic.detail import DetailView

from .models import SCSModel

class SCSDetailView(DetailView):

# specify the model to use

model = SCSModel

Now create a url path to map the view. In scsdbapp/urls.py,

from django.urls import path

# importing views from views..py

from .views import SCSDetailView

urlpatterns = [

# <pk> is identification for id field,

# slug can also be used

path('<pk>/', SCSDetailView.as\_view()),

]

Create template

Create a template in templates/scsdbapp/SCSModel\_detail.html,

<h1>{{ object.title }}</h1>

<p>{{ object.description }}</p>

### Manipulate Context Data in DetailView

By default DetailView will only display fields of a table. If one wants to modify this context data before sending it to template or add some extra field, context data can be overriden.  
In scsdbapp/views.py,

from django.views.generic.detail import DetailView

from .models import SCSModel

class SCSDetailView(DetailView):

# specify the model to use

model = SCSModel

# override context data

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

context = super(SCSDetailView,

self).get\_context\_data(\*args, \*\*kwargs)

# add extra field

context["category"] = "MISC"

return context

In scsdbapp/templates/SCSModel\_detail.html,

<h1>{{ object.title }}</h1>

<p>{{ object.description }}</p>

<p>{{ category }}</p>

**Createview – Class Based Views Django**

Create View refers to a view (logic) to create an instance of a table in the database. We have already discussed basics of Create View in [Create View – Function based Views Django](https://www.geeksforgeeks.org/create-view-function-based-views-django/). Class-based views provide an alternative way to implement views as Python objects instead of functions. They do not replace function-based views, but have certain differences and advantages when compared to function-based views:

* Organization of code related to specific HTTP methods (GET, POST, etc.) can be addressed by separate methods instead of conditional branching.
* Object oriented techniques such as mixins (multiple inheritance) can be used to factor code into reusable components.

Code of models.py

# import the standard Django Model

# from built-in library

from django.db import models

# declare a new model with a name "GeeksModel"

class SCSModel(models.Model):

# fields of the model

title = models.CharField(max\_length = 200)

description = models.TextField()

# renames the instances of the model

# with their title name

def \_\_str\_\_(self):

return self.title

Python manage.py [makemigrations](https://www.geeksforgeeks.org/django-app-model-python-manage-py-makemigrations-command/)

Python manage.py [migrate](https://www.geeksforgeeks.org/django-manage-py-migrate-command-python/)

Class Based Views automatically setup everything from A to Z. One just needs to specify which model to create Create View for and the fields. Then Class based CreateView will automatically try to find a template in app\_name/modelname\_form.html. In our case it is scsapp/templates/scsapp/scsmodel\_form.html. Let’s create our class based view. In scsapp/views.py,

from django.views.generic.edit import CreateView

from .models import GeeksModel

class SCSCreate(CreateView):

# specify the model for create view

model = ScsModel

# specify the fields to be displayed

fields = ['title', 'description']

Now create a url path to map the view. In scsapp/urls.py,

from django.urls import path

# importing views from views..py

from .views import GeeksCreate

urlpatterns = [

path('',ScsCreate.as\_view() ),

]

Create a template in templates/scsapp/scsmodel\_form.html,

<form method="POST" enctype="multipart/form-data">

<!-- Security token -->

{% csrf\_token %}

<!-- Using the formset -->

{{ form.as\_p }}

<input type="submit" value="Submit">

</form>

Verify record using details view or createview

**ListView – Class Based Views Django**

List View refers to a view (logic) to display multiple instances of a table in the database. We have already discussed basics of List View in [List View – Function based Views Django](https://www.geeksforgeeks.org/list-view-function-based-views-django/). Class-based views provide an alternative way to implement views as Python objects instead of functions. They do not replace function-based views, but have certain differences and advantages when compared to function-based views:

* Organization of code related to specific HTTP methods (GET, POST, etc.) can be addressed by separate methods instead of conditional branching.
* Object oriented techniques such as mixins (multiple inheritance) can be used to factor code into reusable components.

Modes.py

# import the standard Django Model

# from built-in library

from django.db import models

# declare a new model with a name "GeeksModel"

class ScsModel(models.Model):

# fields of the model

title = models.CharField(max\_length = 200)

description = models.TextField()

# renames the instances of the model

# with their title name

def \_\_str\_\_(self):

return self.title

Python manage.py [makemigrations](https://www.geeksforgeeks.org/django-app-model-python-manage-py-makemigrations-command/)

Python manage.py [migrate](https://www.geeksforgeeks.org/django-manage-py-migrate-command-python/)

Python manage.py shell

from scsapp.models import ScsModel

>>> ScsModel.objects.create(

title="title1",

description="description1").save()

>>> ScsModel.objects.create(

title="title2",

description="description2").save()

>>> ScsModel.objects.create(

title="title2",

description="description2").save()

Class Based Views automatically setup everything from A to Z. One just needs to specify which model to create ListView for, then Class based ListView will automatically try to find a template in app\_name/modelname\_list.html. In our case it is scsapp/templates/scsapp/scssmodel\_list.html. Let’s create our class based view. In scsapp/views.py,

from django.views.generic.list import ListView

from .models import ScsModel

class SCSList(ListView):

# specify the model for list view

model = GeeksModel

Now create a url path to map the view. In scapp/urls.py,

from django.urls import path

# importing views from views..py

from .views import GeeksList

urlpatterns = [

path('',ScsList.as\_view()),

]

Create a template in templates/scs/scsmodel\_list.html,

<ul>

<!-- Iterate over object\_list -->

{% for object in object\_list %}

<!-- Display Objects -->

<li>{{ object.title }}</li>

<li>{{ object.description }}</li>

<hr/>

<!-- If objet\_list is empty -->

{% empty %}

<li>No objects yet.</li>

{% endfor %}

</ul>

### Manipulate Queryset in ListView

By default ListView will display all instances of a table in the order they were created. If one wants to modify the sequence of these instances or the ordering, get\_queryset method need to be overriden.  
In scsapp/views.py,

from django.views.generic.list import ListView

from .models import GeeksModel

class SCSList(ListView):

# specify the model for list view

model = ScsModel

def get\_queryset(self, \*args, \*\*kwargs):

qs = super(GeeksList, self).get\_queryset(\*args, \*\*kwargs)

qs = qs.order\_by("-id")

return qs