Class Based Generic Views Django (Create, Retrieve, Update, Delete)

Django is a Python-based web framework that allows you to quickly create web applications. It has built-in admin interface which makes easy to work with it. It is often called **Batteries included framework** because it provides built-in facilities for every functionality. Class Based Generic Views are advanced set of Built-in views which are used for implementation of selective view strategies such as Create, Retrieve, Update, Delete. Class based views simplify the use by separating GET, POST requests for a view. 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.

This article revolves around complete implementation of **Class Based Views in Django** (Create, Retrieve, Update, Delete). Let’s discuss what actually CRUD means,

[**CreateView**](https://www.geeksforgeeks.org/class-based-generic-views-django-create-retrieve-update-delete/#createview) – create or add new entries in a table in the database.  
[**Retrieve Views**](https://www.geeksforgeeks.org/class-based-generic-views-django-create-retrieve-update-delete/#retrieveview) – read, retrieve, search, or view existing entries as a list([**ListView**](https://www.geeksforgeeks.org/class-based-generic-views-django-create-retrieve-update-delete/#listview)) or retrieve a particular entry in detail ([**DetailView**](https://www.geeksforgeeks.org/class-based-generic-views-django-create-retrieve-update-delete/#detailview))  
[**UpdateView**](https://www.geeksforgeeks.org/class-based-generic-views-django-create-retrieve-update-delete/#updateview) – update or edit existing entries in a table in the database  
[**DeleteView**](https://www.geeksforgeeks.org/class-based-generic-views-django-create-retrieve-update-delete/#deleteview) – delete, deactivate, or remove existing entries in a table in the database  
[**FormView**](https://www.geeksforgeeks.org/class-based-generic-views-django-create-retrieve-update-delete/#formview) – render a form to template and handle data entered by user

**Django CRUD (Create, Retrieve, Update, Delete) Class Based Views**

Illustration of **How to create and use CRUD views** using an Example. Consider a project named geeksforgeeks having an app named geeks.

*Refer to the following articles to check how to create a project and an app in Django.*

* [*How to Create a Basic Project using MVT in Django?*](https://www.geeksforgeeks.org/how-to-create-a-basic-project-using-mvt-in-django/)
* [*How to Create an App in Django ?*](https://www.geeksforgeeks.org/how-to-create-an-app-in-django/)

After you have a project and an app, let’s create a model of which we will be creating instances through our view. In geeks/models.py,

filter\_none

brightness\_4

|  |
| --- |
| # import the standard Django Model  # from built-in library  from django.db import models    # declare a new model with a name "GeeksModel"  class GeeksModel(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 |

After creating this model, we need to run two commands in order to create Database for the same.

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/)

Now we will create a Django ModelForm for this model. Refer this article for more on modelform – [Django ModelForm – Create form from Models](https://geeksforgeeks.org/django-modelform-create-form-from-models/). create a file forms.py in geeks folder,

filter\_none

brightness\_4

|  |
| --- |
| from django import forms  from .models import GeeksModel      # creating a form  class GeeksForm(forms.ModelForm):        # create meta class      class Meta:          # specify model to be used          model = GeeksModel            # specify fields to be used          fields = [              "title",              "description",          ] |

**Using Class Based Views**

At its core, a class-based view allows you to respond to different HTTP request methods with different class instance methods, instead of with conditionally branching code inside a single view function.

So where the code to handle HTTP GET in a view function would look something like:

filter\_none

brightness\_4

|  |
| --- |
| from django.http import HttpResponse    def my\_view(request):      if request.method == 'GET':          # <view logic>          return HttpResponse('result') |

In a class-based view, this would become:

filter\_none

brightness\_4

|  |
| --- |
| from django.http import HttpResponse  from django.views import View    class MyView(View):      def get(self, request):          # <view logic>          return HttpResponse('result') |

Similarly in urls.py, one needs to use as\_view() method to diffrentiate a class based view from function based view.

filter\_none

brightness\_4

|  |
| --- |
| # urls.py  from django.urls import path  from myapp.views import MyView    urlpatterns = [        path('about/', MyView.as\_view()),    ] |

**CreateView**

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. 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 geeks/templates/geeks/geeksmodel\_form.html. Let’s create our class based view. In geeks/views.py,

filter\_none

brightness\_4

|  |
| --- |
| from django.views.generic.edit import CreateView  from .models import GeeksModel    class GeeksCreate(CreateView):        # specify the model for create view      model = GeeksModel        # specify the fields to be displayed        fields = ['title', 'description'] |

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

filter\_none

brightness\_4

|  |
| --- |
| from django.urls import path    # importing views from views..py  from .views import GeeksCreate  urlpatterns = [      path('', GeeksCreate.as\_view() ),  ] |

Create a template in templates/geeks/geeksmodel\_form.html,

filter\_none

brightness\_4

|  |
| --- |
| <form method="POST" enctype="multipart/form-data">        <!-- Security token -->      {% csrf\_token %}        <!-- Using the formset -->      {{ form.as\_p }}        <input type="submit" value="Submit">  </form> |

Let’s check what is there on <http://localhost:8000/>  
  
To check complete implementation of Class based CreateView, visit [Createview – Class Based Views Django](https://www.geeksforgeeks.org/createview-class-based-views-django/).

**Retrieve Views**

**ListView**

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 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 geeks/templates/geeks/geeksmodel\_list.html. Let’s create our class based view. In geeks/views.py,

filter\_none

brightness\_4

|  |
| --- |
| from django.views.generic.list import ListView  from .models import GeeksModel    class GeeksList(ListView):        # specify the model for list view      model = GeeksModel |

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

filter\_none

brightness\_4

|  |
| --- |
| from django.urls import path    # importing views from views..py  from .views import GeeksList  urlpatterns = [      path('', GeeksList.as\_view()),  ] |

Create a template in templates/geeks/geeksmodel\_list.html,

filter\_none

brightness\_4

|  |
| --- |
| <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> |

Let’s check what is there on <http://localhost:8000/>  
  
To check complete implementation of Class based ListView, visit [ListView – Class Based Views Django](https://www.geeksforgeeks.org/listview-class-based-views-django/)

**DetailView**

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 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/geeks/geeksmodel\_detail.html. Let’s create our class based view. In geeks/views.py,

filter\_none

brightness\_4

|  |
| --- |
| from django.views.generic.detail import DetailView    from .models import GeeksModel    class GeeksDetailView(DetailView):      # specify the model to use      model = GeeksModel |

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

filter\_none

brightness\_4

|  |
| --- |
| from django.urls import path    # importing views from views..py  from .views import GeeksDetailView  urlpatterns = [      # <pk> is identification for id field,      # slug can also be used      path('<pk>/', GeeksDetailView.as\_view()),  ] |

Create a template in templates/geeks/geeksmodel\_detail.html,

filter\_none

brightness\_4

|  |
| --- |
| <h1>{{ object.title }}</h1>  <p>{{ object.description }}</p> |

Let’s check what is there on [http://localhost:8000/1/](http://localhost:8000/1)  
  
To check complete implementation of Class based DetailView, visit [DetailView – Class Based Views Django](https://www.geeksforgeeks.org/detailview-class-based-views-django/)

**UpdateView**

UpdateView refers to a view (logic) to update a particular instance of a table from the database with some extra details. It is used to update enteries in the database for example, updating an article at geeksforgeeks. We have already discussed basics of Update View in [Update View – Function based Views Django](https://www.geeksforgeeks.org/update-view-function-based-views-django/). Class Based Views automatically setup everything from A to Z. One just needs to specify which model to create UpdateView for, then Class based UpdateView will automatically try to find a template in app\_name/modelname\_form.html. In our case it is geeks/templates/geeks/geeksmodel\_form.html. Let’s create our class based view. In geeks/views.py,

filter\_none

brightness\_4

|  |
| --- |
| # import generic UpdateView  from django.views.generic.edit import UpdateView    # Relative import of GeeksModel  from .models import GeeksModel    class GeeksUpdateView(UpdateView):      # specify the model you want to use      model = GeeksModel        # specify the fields      fields = [          "title",          "description"      ]        # can specify success url      # url to redirect after sucessfully      # updating details      success\_url ="/" |

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

filter\_none

brightness\_4

|  |
| --- |
| from django.urls import path    # importing views from views..py  from .views import GeeksUpdateView  urlpatterns = [      # <pk> is identification for id field,      # <slug> can also be used      path('<pk>/update', GeeksUpdateView.as\_view()),  ] |

Create a template in templates/geeks/geeksmodel\_form.html,

filter\_none

brightness\_4

|  |
| --- |
| <form method="post">      {% csrf\_token %}      {{ form.as\_p }}      <input type="submit" value="Save">  </form> |

Let’s check what is there on [http://localhost:8000/1/update/](http://localhost:8000/1/update)

To check complete implementation of Class based UpdateView, visit [UpdateView – Class Based Views Django](https://www.geeksforgeeks.org/updateview-class-based-views-django/).

**DeleteView**

Delete View refers to a view (logic) to delete a particular instance of a table from the database. It is used to delete enteries in the database for example, deleting an article at geeksforgeeks. We have already discussed basics of Delete View in [Delete View – Function based Views Django](https://www.geeksforgeeks.org/delete-view-function-based-views-django/). Class Based Views automatically setup everything from A to Z. One just needs to specify which model to create DeleteView for, then Class based DeleteViewde will automatically try to find a template in app\_name/modelname\_confirm\_delete.html. In our case it is geeks/templates/geeks/geeksmodel\_confirm\_delete.html. Let’s create our class based view. In geeks/views.py,

filter\_none

brightness\_4

|  |
| --- |
| # import generic UpdateView  from django.views.generic.edit import DeleteView    # Relative import of GeeksModel  from .models import GeeksModel    class GeeksDeleteView(DeleteView):      # specify the model you want to use      model = GeeksModel        # can specify success url      # url to redirect after sucessfully      # deleting object      success\_url ="/" |

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

filter\_none

brightness\_4

|  |
| --- |
| from django.urls import path    # importing views from views..py  from .views import GeeksDeleteView  urlpatterns = [      # <pk> is identification for id field,      # slug can also be used      path('<pk>/delete/', GeeksDeleteView.as\_view()),  ] |

Create a template in templates/geeks/geeksmodel\_confirm\_delete.html,

filter\_none

brightness\_4

|  |
| --- |
| <form method="post">{% csrf\_token %}      <p>Are you sure you want to delete "{{ object }}"?</p>      <input type="submit" value="Confirm">  </form> |

Let’s check what is there on <http://localhost:8000/1/delete>  
.  
To check complete implementation of Class based DeleteView, visit [DeleteView – Class Based Views Django](https://www.geeksforgeeks.org/deleteview-class-based-views-django/)

**FormView**

FormView refers to a view (logic) to display and verify a Django Form. For example a form to register users at geeksforgeeks. Class Based Views automatically setup everything from A to Z. One just needs to specify which form to create FormView for and template\_name, then Class based FormView will automatically render that form. Let’s create our class based view. In geeks/views.py,

filter\_none

brightness\_4

|  |
| --- |
| # import generic FormView  from django.views.generic.edit import FormView    # Relative import of GeeksForm  from .forms import GeeksForm    class GeeksFormView(FormView):      # specify the Form you want to use      form\_class = GeeksForm        # sepcify name of template      template\_name = "geeks / geeksmodel\_form.html"        # can specify success url      # url to redirect after sucessfully      # updating details      success\_url ="/thanks/" |

Create a template for this view in geeks/geeksmodel\_form.html,

filter\_none

brightness\_4

|  |
| --- |
| <form method="post">      {% csrf\_token %}      {{ form.as\_p }}      <input type="submit" value="Save">  </form> |

Map a url to this view in geeks/urls.py,

filter\_none

brightness\_4

|  |
| --- |
| from django.urls import path    # importing views from views..py  from .views import GeeksFormView  urlpatterns = [      path('', GeeksFormView.as\_view()),  ] |

Now visit <http://127.0.0.1:8000/>,  
  
To check complete implementation of Class based FormView, visit [FormView – Class Based Views Django](https://www.geeksforgeeks.org/formview-class-based-views-django/)

**Recommended Posts:**

* [Django CRUD (Create, Retrieve, Update, Delete) Function Based Views](https://www.geeksforgeeks.org/django-crud-create-retrieve-update-delete-function-based-views/?ref=rp)
* [Update View - Function based Views Django](https://www.geeksforgeeks.org/update-view-function-based-views-django/?ref=rp)
* [Delete View - Function based Views Django](https://www.geeksforgeeks.org/delete-view-function-based-views-django/?ref=rp)
* [Create View - Function based Views Django](https://www.geeksforgeeks.org/create-view-function-based-views-django/?ref=rp)
* [DetailView - Class Based Views Django](https://www.geeksforgeeks.org/detailview-class-based-views-django/?ref=rp)
* [Createview - Class Based Views Django](https://www.geeksforgeeks.org/createview-class-based-views-django/?ref=rp)
* [FormView - Class Based Views Django](https://www.geeksforgeeks.org/formview-class-based-views-django/?ref=rp)
* [ListView - Class Based Views Django](https://www.geeksforgeeks.org/listview-class-based-views-django/?ref=rp)
* [UpdateView - Class Based Views Django](https://www.geeksforgeeks.org/updateview-class-based-views-django/?ref=rp)
* [DeleteView - Class Based Views Django](https://www.geeksforgeeks.org/deleteview-class-based-views-django/?ref=rp)
* [Detail View - Function based Views Django](https://www.geeksforgeeks.org/detail-view-function-based-views-django/?ref=rp)
* [List View - Function based Views Django](https://www.geeksforgeeks.org/list-view-function-based-views-django/?ref=rp)
* [Views In Django | Python](https://www.geeksforgeeks.org/views-in-django-python/?ref=rp)
* [Render a HTML Template as Response - Django Views](https://www.geeksforgeeks.org/render-a-html-template-as-response-django-views/?ref=rp)
* [How to Create an App in Django ?](https://www.geeksforgeeks.org/how-to-create-an-app-in-django/?ref=rp)

[**NaveenArora**](https://auth.geeksforgeeks.org/user/NaveenArora/articles)

Check out this Author's [contributed articles](https://auth.geeksforgeeks.org/user/NaveenArora/articles).

If you like GeeksforGeeks and would like to contribute, you can also write an article using [contribute.geeksforgeeks.org](https://contribute.geeksforgeeks.org/) or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please Improve this article if you find anything incorrect by clicking on the "Improve Article" button below.