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

### Field types

The generated Form class will have a form field for every model field specified, in the order specified in the fields attribute. Each model field has a corresponding default form field. For example, a CharField on a model is represented as a CharField on a form. A model ManyToManyField is represented as a MultipleChoiceField. Here is the full list of conversions:

| Model Field | Form Field |
| --- | --- |
| AutoField | Not represented in the form |
| BigAutoField | Not represented in the form |
| BigIntegerField | IntegerField with min\_value set to -9223372036854775808 and max\_value set to 9223372036854775807. |
| BinaryField | CharField, if editable is set to True on the model field, otherwise not represented in the form. |
| BooleanField | BooleanField, or NullBooleanField if null=True. |
| CharField | CharField with max\_length set to the model field’s max\_length and empty\_value set to None if null=True. |
| DateField | DateField |
| DateTimeField | DateTimeField |
| DecimalField | DecimalField |
| DurationField | DurationField |
| EmailField | EmailField |
| FileField | FileField |
| FilePathField | FilePathField |
| FloatField | FloatField |
| ForeignKey | ModelChoiceField |
| ImageField | ImageField |
| IntegerField | IntegerField |
| IPAddressField | IPAddressField |
| GenericIPAddressField | GenericIPAddressField |
| ManyToManyField | ModelMultipleChoiceField |
| NullBooleanField | NullBooleanField |
| PositiveIntegerField | IntegerField |
| PositiveSmallIntegerField | IntegerField |
| SlugField | SlugField |
| SmallAutoField | Not represented in the form |
| TextField | CharField with widget=forms.Textarea |
| TimeField | TimeField |
| URLField | URLField |

**Django ModelForm** is a class that is used to directly convert a model into a Django form. If you’re building a database-driven app, chances are you’ll have forms that map closely to Django models. For example, a User Registration model and form would have same quality and quantity of model fields and form fields. So instead of creating a redundant code to first create a form and then map it to the model in a view, we can directly use ModelForm. It takes as argument the name of the model and converts it into a Django Form. Not only this, ModelForm offers a lot of methods and features which automate the entire process and help remove code redundancy.

Models.py

|  |
| --- |
| from django.db import models  class GeeksModel(models.Model):  # fields of the model  title = models.CharField(max\_length = 200)  description = models.TextField()  last\_modified = models.DateTimeField(auto\_now\_add = True)  img = models.ImageField(upload\_to = "images/")  # renames the instances of the model  # with their title name  def \_\_str\_\_(self):  return self.title |

Now, run following commands to create the model,

Python manage.py makemigrations

Python manage.py migrate

**Forms.py**

|  |
| --- |
| from django import forms  # import GeeksModel from models.py  from .models import GeeksModel  # create a ModelForm  class GeeksForm(forms.ModelForm):  # specify the name of model to use  class Meta:  model = GeeksModel  fields = "\_\_all\_\_" |

This form takes two arguments **fields** or **exclude**.

* **fields –** It is strongly recommended that you explicitly set all fields that should be edited in the form using the fields attribute. Failure to do so can easily lead to security problems when a form unexpectedly allows a user to set certain fields, especially when new fields are added to a model. Depending on how the form is rendered, the problem may not even be visible on the web page. Set the fields attribute to the special value **‘\_\_all\_\_’** to indicate that all fields in the model should be used.
* **exclude –**Set the exclude attribute of the ModelForm’s inner Meta class to a list of fields to be excluded from the form.  
  For example:

class PartialAuthorForm(ModelForm):

class Meta:

model = Author

exclude = ['title']

Finally, to complete our MVT structure, create a view that would render the form and directly save it to the database. In **geeks/views.py,**

|  |
| --- |
| from django.shortcuts import render  from .forms import GeeksForm  def home\_view(request):  context ={}  # create object of form  form = GeeksForm(request.POST or None, request.FILES or None)  # check if form data is valid  if form.is\_valid():  # save the form data to model  form.save()  context['form']= form  return render(request, "home.html", context) |

### Create View

Create View refers to a view (logic) to create an instance of a table in the database. It is just like taking an input from a user and storing it in a specified table.   
In geeks/views.py, 

|  |
| --- |
| from django.shortcuts import render  # relative import of forms  from .models import GeeksModel  from .forms import GeeksForm  def create\_view(request):  # dictionary for initial data with  # field names as keys  context ={}  # add the dictionary during initialization  form = GeeksForm(request.POST or None)  if form.is\_valid():  form.save()  context['form']= form  return render(request, "create\_view.html", context) |

**Create a template in templates/create\_view.html,**

* html

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

### Retrieve View

Retrieve view is basically divided into two types of views Detail View and List View. 

#### List View

List View refers to a view (logic) to list all or particular instances of a table from the database in a particular order. It is used to display multiple types of data on a single page or view, for example, products on an eCommerce page.   
In geeks/views.py, 

* Python3

|  |
| --- |
| from django.shortcuts import render  # relative import of forms  from .models import GeeksModel  def list\_view(request):  # dictionary for initial data with  # field names as keys  context ={}  # add the dictionary during initialization  context["dataset"] = GeeksModel.objects.all()  return render(request, "list\_view.html", context) |

**Create a template in templates/list\_view.html,**

* html

|  |
| --- |
| <div class="main">  {% for data in dataset %}.  {{ data.title }}<br/>  {{ data.description }}<br/>  <hr/>  {% endfor %}  </div> |

#### Detail View

Detail View refers to a view (logic) to display a particular instance of a table from the database with all the necessary details. It is used to display multiple types of data on a single page or view, for example, profile of a user.   
In geeks/views.py, 

* Python3

|  |
| --- |
| from django.urls import path  # importing views from views..py  from .views import detail\_view  urlpatterns = [  path('<id>', detail\_view ),  ] |

Let’s create a view and template for the same. In geeks/views.py,

* Python3

|  |
| --- |
| from django.shortcuts import render  # relative import of forms  from .models import GeeksModel  # pass id attribute from urls  def detail\_view(request, id):  # dictionary for initial data with  # field names as keys  context ={}  # add the dictionary during initialization  context["data"] = GeeksModel.objects.get(id = id)  return render(request, "detail\_view.html", context) |

Create a template in templates/Detail\_view.html, 

* html

|  |
| --- |
| <div class="main">  <!-- Specify fields to be displayed -->  {{ data.title }}<br/>  {{ data.description }}<br/>  </div> |

### Update View

Update View 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 entries in the database for example, updating an article at geeksforgeeks.   
In geeks/views.py,

* Python3

|  |
| --- |
| from django.shortcuts import (get\_object\_or\_404,  render,  HttpResponseRedirect)  # relative import of forms  from .models import GeeksModel  from .forms import GeeksForm  # after updating it will redirect to detail\_View  def detail\_view(request, id):  # dictionary for initial data with  # field names as keys  context ={}  # add the dictionary during initialization  context["data"] = GeeksModel.objects.get(id = id)  return render(request, "detail\_view.html", context)  # update view for details  def update\_view(request, id):  # dictionary for initial data with  # field names as keys  context ={}  # fetch the object related to passed id  obj = get\_object\_or\_404(GeeksModel, id = id)  # pass the object as instance in form  form = GeeksForm(request.POST or None, instance = obj)  # save the data from the form and  # redirect to detail\_view  if form.is\_valid():  form.save()  return HttpResponseRedirect("/"+id)  # add form dictionary to context  context["form"] = form  return render(request, "update\_view.html", context) |

Now create following templates in templates folder,   
In geeks/templates/update\_view.html,

* html

|  |
| --- |
| <div class="main">  <!-- Create a Form -->  <form method="POST">  <!-- Security token by Django -->  {% csrf\_token %}  <!-- form as paragraph -->  {{ form.as\_p }}  <input type="submit" value="Update">  </form>  </div> |

In geeks/templates/detail\_view.html, 

* html

|  |
| --- |
| <div class="main">  <!-- Display attributes of instance -->  {{ data.title }} <br/>  {{ data.description }}  </div> |

### Delete View

Delete View refers to a view (logic) to delete a particular instance of a table from the database. It is used to delete entries in the database for example, deleting an article at geeksforgeeks.   
In geeks/views.py 

* Python3

|  |
| --- |
| from django.shortcuts import (get\_object\_or\_404,  render,  HttpResponseRedirect)  from .models import GeeksModel  # delete view for details  def delete\_view(request, id):  # dictionary for initial data with  # field names as keys  context ={}  # fetch the object related to passed id  obj = get\_object\_or\_404(GeeksModel, id = id)  if request.method =="POST":  # delete object  obj.delete()  # after deleting redirect to  # home page  return HttpResponseRedirect("/")  return render(request, "delete\_view.html", context) |

Now a url mapping to this view with a regular expression of id,   
In geeks/urls.py 

* Python3

|  |
| --- |
| from django.urls import path  # importing views from views..py  from .views import delete\_view  urlpatterns = [  path('<id>/delete', delete\_view ),  ] |

Template for delete view includes a simple form confirming whether user wants to delete the instance or not. In geeks/templates/delete\_view.html, 

* html

|  |
| --- |
| <div class="main">  <!-- Create a Form -->  <form method="POST">  <!-- Security token by Django -->  {% csrf\_token %}  Are you want to delete this item ?  <input type="submit" value="Yes" />  <a href="/">Cancel </a>  </form>  </div> |