Steps for Database driven application (CRUD) in django :

1. **Create a project CRUD_DEMO.**
2. **Create app “blog” inside your project.**

3. **Open blog/models.py and enter below code :**

```python
# import the standard Django Model
# from built-in library
from django.db import models

# declare a new model with a name "BlogModel"
class BlogModel(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
```

6. Add the name of the application in INSTALLED_APPS section.

7. Open terminal and run “python3.6 manage.py makemigrations”

8. Open terminal and run “python3.6 manage.py migrate”.

9. Go to crud_demo/urls.py and enter below code :

```python
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('blog/', include('blog.urls')),
]
```

10. Create a file blog/forms.py and write the below code :

```python
from django import forms
from .models import BlogModel
```

```python
# creating a form
class BlogForms(forms.ModelForm):

    # create meta class
    class Meta:
        # specify model to be used
        model = BlogModel

        # specify fields to be used
        fields = [
            "title",
            "description",
        ]
```

11. **Create blog/urls.py file and write the below code :**

```python
from django.urls import path

from . import views

urlpatterns = [
    path('', views.create_view, name='create_view'),
    path('list', views.list_view, name='list_view'),
    path('<id>',views.detail_view, name='detail_view'),
    path('<id>/update',views.update_view, name='update_view'),
    path('<id>/delete',views.delete_view, name='delete_view'),
]
```

12. **Open blog/views.py and write the below code :**

```python
from django.shortcuts import render
from django.shortcuts import (get_object_or_404,
                              render,
                              HttpResponseRedirect)


# relative import of forms
from .models import BlogModel
from .forms import BlogForms


def create_view(request):
    # dictionary for initial data with
    # field names as keys
    context ={}

    # add the dictionary during initialization
    form = BlogForms(request.POST or None)
    if form.is_valid():
        form.save()
    context['form']= form
    return render(request, "create_view.html",context)

def list_view(request):
    # dictionary for initial data with
    # field names as keys
    context ={}

    # add the dictionary during initialization
    context["dataset"] = BlogModel.objects.all()

    return render(request, "list_view.html", context)

# 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"] = BlogModel.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(BlogModel, id = id)

    # pass the object as instance in form
    form = BlogForms(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("/blog/"+id)

    # add form dictionary to context
    context["form"] = form

    return render(request, "update_view.html", context)

# 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(BlogModel, id = id)


    if request.method =="POST":
        # delete object
        obj.delete()
        # after deleting redirect to
        # home page
        return HttpResponseRedirect("/blog/list")

    return render(request, "delete_view.html", context)
```

13. Create template folder in the main project(root) directory and create below html files :

Create_view.html

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

    <!-- Security token -->
    {% csrf_token %}

    <!-- Using the formset -->
    {{ form.as_p }}

    <input type="submit" value="Submit">
</form>
```

Detail_view.html

```
<div class="main">

    <!-- Specify fields to be displayed -->
    {{ data.title }}<br/>
    {{ data.description }}<br/>

</div>
```

List_view.html

```
<div class="main">
```

```
    {% for data in dataset %}.

    {{ data.title }}<br/>
    {{ data.description }}<br/>
    <hr/>

    {% endfor %}

</div>
```

Update_view.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>
```

Delete_view.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>
```

Test your application with below URL

127.0.0.1:8000/blog – Create
127.0.0.1:8000/blog/list – List of all items
127.0.0.1:8000/blog/<<id>> - Details of item with <<id>>
127.0.0.1:8000/blog/<<id>>/update – For updating item with <<id>>
127.0.0.1:8000/blog/<<id>>/delete – Delete item with <<id>>

Djano models :

https://www.geeksforgeeks.org/django-model-data-types-and-fields-list/

Makemigrations : Responsible for creating new migrations based on the changes you have made to your models.

Migrate : Responsible for applying and unapplying migrations. Creating the actual tables in the database.

Forms.py : In Django, the "forms.py" file is used to create and handle forms. It is used to define the fields for the form, their types, and any validation rules that need to be applied. These forms can then be used in views to handle user input and to display the form in templates. The forms can also be used to validate user input and to perform any necessary data processing before it is saved to the database. Additionally, it can be used as a way to handle forms in the frontend of a web application. It is connected with model.

Request.post or none :

In Django, the `request.POST` variable is a dictionary-like object that contains all the data that is sent in a POST request. It is often used in views to handle forms that are submitted by the user.

The `None` value is used in conjunction with the `request.POST` variable to create a form instance. When a form is created, it can be initialized with data from the `request.POST` object, if it is available. If the form is being displayed for the first time and no data has been submitted yet, then `request.POST` will be empty, and passing `None` as an argument to the form will initialize an empty form.

Form.is_valid()

In Django, the `form.is_valid()` method is used to check if the data submitted in a form is valid according to the validation rules defined in the form's fields. When this method is called, it will run the form's cleaning and validation functions for each field and check that all the fields pass validation.

Form.save()

In Django, the `form.save()` method is used to save the data from a form to the database. It is typically used in views after the form has been validated using the `form.is_valid()` method.

{{ form.as_p }}

In Django, the `{{ form.as_p }}` template tag is used to render a form's fields as a series of paragraphs. It is a shortcut for calling `{{ form.as_p }}` on each field individually.

Model.objects.all()

In Django, the `objects` attribute of a model is a manager object that allows you to interact with the database using the model's corresponding table. The `all()` method is a method on the manager object that returns a queryset containing all the objects in the database table for that model.

Model.objects.get()

In Django, the `objects` attribute of a model is a manager object that allows you to interact with the database using the model's corresponding table. The `get()` method is a method on the manager object that allows you to retrieve a single object from the database that matches the given lookup parameters.

`get_object_or_404`()

Is a function in Django that is used to retrieve a single object based on the provided lookup parameters. If the object is not found, it raises a Http404 exception.

Modelobject.delete()

In Django, the `delete()` method is used to delete an object from the database. This method is a part of the Django's Model class and can be called on any instance of a model.