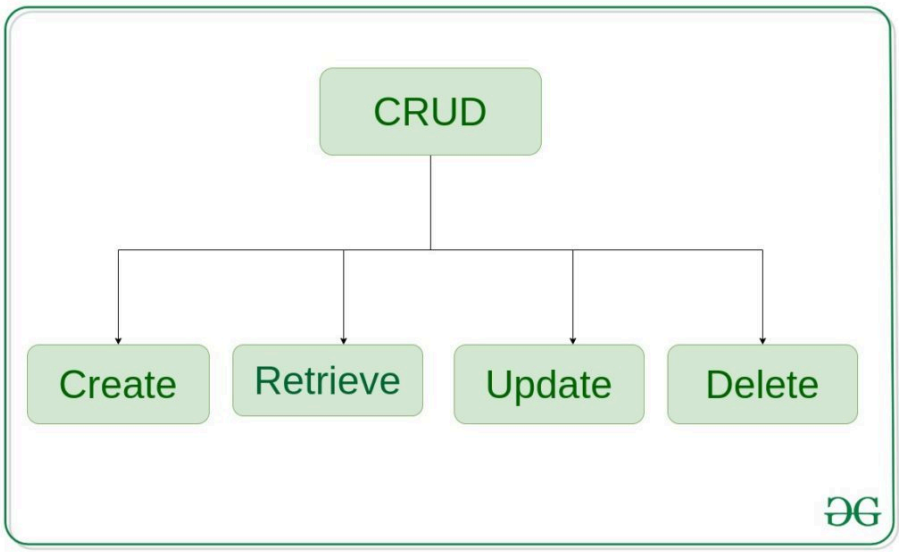
Django Class Based Views

Last Updated : 23 Sep, 2024

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



CreateView – create or add new entries in a table in the database.
Retrieve Views – read, retrieve, search, or view existing entries as a list(**ListView**) or retrieve a particular entry in detail (**DetailView**)
UpdateView – update or edit existing entries in a table in the database
DeleteView – delete, deactivate, or remove existing entries in a table in the database
FormView – render a form to template and handle data entered by user

Django Class Based Views CRUD Operations

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?*
- *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,

Python3

```python
# 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
Python manage.py migrate
```

Now we will create a Django ModelForm for this model. Refer this article for more on modelform – Django ModelForm – Create form from Models. create a file forms.py in geeks folder,

Python3

```python
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:

Python3

```python
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:

Python3

```python
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 differentiate a class based view from function based view.

Python3

```python
# 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,

Python3

```python
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,

Python3

```python
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,

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>
```

Let's check what is there on http://localhost:8000/

To check complete implementation of Class based CreateView, visit 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. 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,

Python3

```python
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,

Python3

```python
from django.urls import path

# importing views from views..py
from .views import GeeksList
urlpatterns = [
    path('', GeeksList.as_view()),
]
```

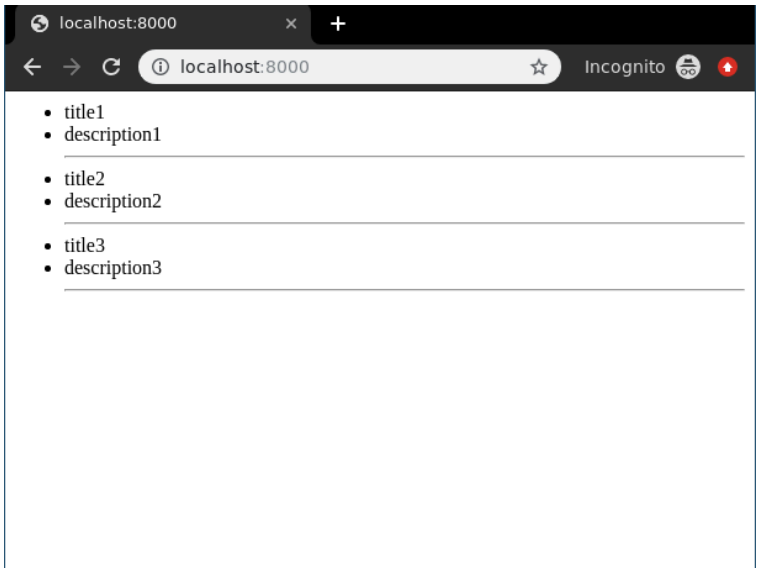
Create a template in templates/geeks/geeksmodel_list.html,

HTML

```html
<ul>
    <!-- Iterate over object_list -->
    {% for object in object_list %}
    <!-- Display Objects -->
    <li>{{ object.title }}</li>
    <li>{{ object.description }}</li>

    <hr/>
    <!-- If object_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

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. 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,

Python3

```python
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,

Python3

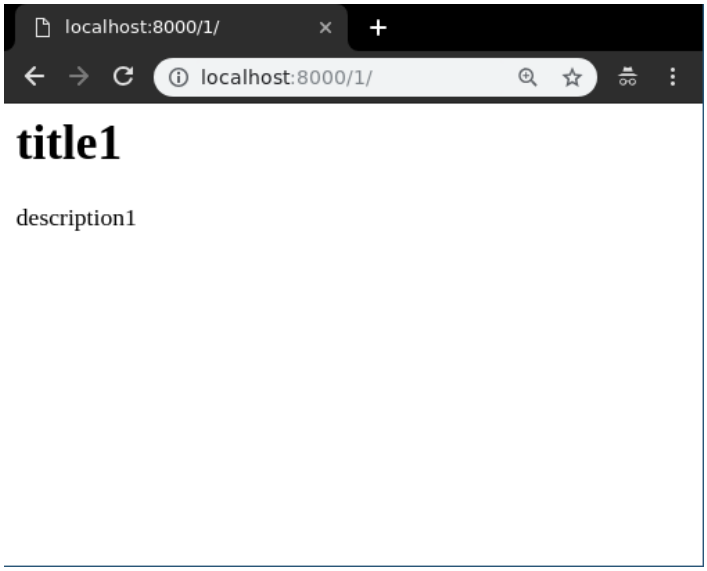
```python
from django.urls import path

# 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,

HTML

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



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

Let's check what is there on http://localhost:8000/1/



To check complete implementation of Class based DetailView, visit 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 entries 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. 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,

Python3

```python
# 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 successfully
    # updating details
    success_url ="/"
```

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

Python3

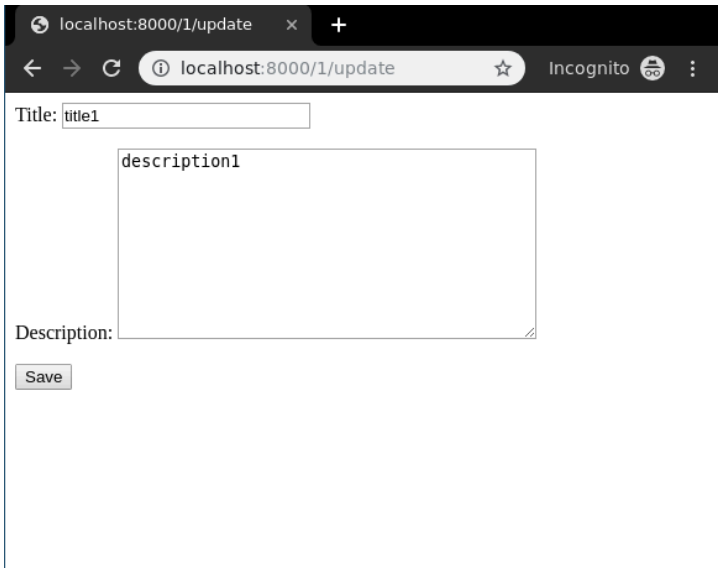
```python
from django.urls import path

# 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,

HTML

```html
<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/



To check complete implementation of Class based UpdateView, visit 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 entries 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. 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,

Python3

```python
# 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 successfully
    # deleting object
    success_url ="/"
```

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

Python3

```python
from django.urls import path

# importing views from views..py
from .views import GeeksDeleteView
```

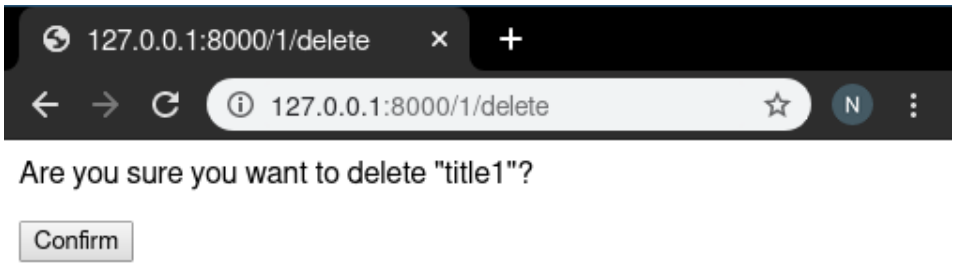
```
urlpatterns = [
    # <pk> is identification for id field,
    # slug can also be used
    path('<pk>/delete/', GeeksDeleteView.as_view()),
]
```

Create a template in templates/geeks/geeksmodel_confirm_delete.html,

HTML

```
<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

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,

Python3

```python
# 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

    # specify name of template
    template_name = "geeks / geeksmodel_form.html"

    # can specify success url
    # url to redirect after successfully
    # updating details
    success_url ="/thanks/"
```

Create a template for this view in geeks/geeksmodel_form.html,

HTML

```html
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Save">
</form>
```

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

Python3

```python
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

Are you ready to elevate your web development skills from foundational knowledge to advanced expertise? Explore our Mastering Django Framework - Beginner to Advanced Course on GeeksforGeeks, designed for aspiring developers and experienced programmers. This comprehensive course covers everything you need to know about Django, from the basics to advanced features. Gain practical experience through **hands-on projects** and real-world applications, mastering essential Django principles and techniques. Whether you're just starting or looking to refine your skills, this course will empower you

to build sophisticated web applications efficiently. Ready to enhance your web development journey? Enroll now and unlock your potential with Django!

Previous Article

Delete View - Function based Views Django

Next Article

Createview - Class Based Views Django

Similar Reads

Class Based vs Function Based Views - Which One is Better to Use in Django?

Django...We all know the popularity of this python framework all over the world. This framework has made life easier for developers. It has become easier for...

7 min read

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 –...

3 min read

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 the basics of List View in List View –...

4 min read

UpdateView - Class Based Views Django

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 entries in the databas...

3 min read

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 –...

3 min read

DeleteView - Class Based Views Django

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 a...

3 min read

FormView - Class Based Views Django

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 provide ...

3 min read

Class based views - Django Rest Framework

Class-based views help in composing reusable bits of behavior. Django REST Framework provides several pre-built views that allow us to reuse common...

12 min read

How to Use permission_required Decorators with Django Class-Based Views

In Django, permissions are used to control access to views and resources. When working with function-based views (FBVs), decorators like @permission_require...

4 min read

Create View - Function based Views Django

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...

3 min read

Article Tags : Django Python Django-views Python Django

Practice Tags : python