7.Class Based View

In Django, views are responsible for processing HTTP requests and returning appropriate HTTP responses.

There are two main approaches to writing views: function-based views (FBVs) and class-based views (CBVs).

1.)Function-Based Views (FBVs):

- FBVs are the traditional way of writing views in Django.
- They are simple Python functions that take an HTTP request as input and return an HTTP response.

Example-1

```
from django.http import HttpResponse

def my_view(request):
    return HttpResponse("Hello, world!")
```

2.)Class-Based Views (CBVs):

Class-Based Views (CBVs) are a fundamental aspect of Django's architecture for organizing and implementing views. Instead of defining a view function directly, you create a class that

inherits from one of Django's built-in view classes

. These classes offer a set of methods that you can override to tailor the behavior of your view to your specific needs.

There are several compelling advantages to adopting class-based views in Django:

1. **Consistent, Object-Oriented Interface:** CBVs provide a consistent and object-oriented approach to structuring your views. By leveraging classes, you can encapsulate related functionality and state within the view, promoting cleaner and more organized code.

1. **Code Reusability:** One of the key benefits of CBVs is their facilitation of code reuse. By encapsulating common behavior within a class, you can easily reuse that behavior across multiple views by inheriting from and extending base view classes. This promotes DRY (Don't Repeat Yourself) principles and reduces redundancy in your codebase.

1. **Encapsulation of Complex Behavior:**

1. CBVs enable you to encapsulate complex behavior within your views, improving readability and maintainability.

1. By leveraging inheritance and method overrides, you can customize the behavior of your views in a modular and organized manner, making it easier to understand and modify the functionality as your project evolves.

In summary, Class-Based Views in Django offer a powerful and flexible mechanism for structuring your views.

By leveraging object-oriented principles and inheritance, CBVs promote code reuse, encapsulation of behavior, and a consistent interface for defining and customizing views in your Django application.

- CBVs are an alternative approach introduced in Django to address some of the limitations of FBVs, particularly in terms of code organization and reuse.

- CBVs are implemented as Python classes instead of functions.
- Django provides a variety of built-in generic class-based views that
- you can use out of the box, such as ListView, DetailView, CreateView, UpdateView, and DeleteView.

Example-2 class-based view:

```
from django.http import HttpResponse

from django.views import View


class MyView(View):

    def get(self, request):

        return HttpResponse("Hello, world!")
```

CBVs offer several advantages over FBVs, including:

Better code organization: CBVs encourage organizing related functionality into classes, making it easier to manage and maintain complex views.

Code reuse: CBVs make it easier to reuse common patterns and behaviors across multiple views by inheriting from and extending base view classes.

Built-in functionality: Django provides a variety of built-in generic CBVs that encapsulate common patterns, such as handling CRUD operations for models.

Both FBVs and CBVs are valid approaches to writing views in Django, and the choice between them ultimately depends on your specific requirements, preferences, and the complexity of your application.

In many cases, you may find yourself using a combination of both FBVs and CBVs, depending on the context and requirements of individual views.

1.get (self, request, *args, **kwargs):

- This method is called when a GET request is made to the view.
- It is used to retrieve and display data, typically for viewing a list of objects.
- commonly used for displaying a list of items, like in a `ListView`.

2. post (self, request, *args, **kwargs):

-
- This method is called when a POST request is made to the view, usually from a form submission.
- It is used to process the data submitted via the form and perform actions like saving data to the database.
- Commonly used for handling form submissions, like in a `CreateView` or `UpdateView`.

```
class TaskListView(View):


  def get(self, request, *args, **kwargs):

    return render(request, 'todo/task_list.html', {

      'tasks': Task.objects.all(),

    })


  def post(self, request, pk, *args, **kwargs):

    task = get_object_or_404(Task, pk=pk)

    form = ConfirmForm(data=request.POST)

    if form.is_valid():

      task.delete()

      return redirect('task-list')
```

```
        return self.get(request, pk)
```