Django Class Based Views Concept

- Django works on the MVT concept we mainly work on two types of views in it they are class-based views and function-based views.
- If you're new to the Django framework then surely you might have been using FBVs (Function Based Views).
- Initially, Django started with the Function Based Views but later Django added the concept of class-based views to avoid the redundancy of code in the boilerplate.
- It is a debate among developers which one is better to use in Django… class-based views or function-based views? Today in this blog we are going to discuss this topic in-depth to get to know the pros and cons of both of the views.
- You can accomplish your task using both of them. Some tasks can be best implemented using CBVs and some of them can be implemented in FBVs.

Django views have mainly three requirements…

1. They are callable. You can write the views either using function-based or class-based. While using CBVs you inherit the method **as_view()** that uses the **dispatch()** method to call the method that is appropriate depending on the HTTP verb (get, post), etc.
2. As a first positional argument, Django views should accept HttpRequest.
3. It should return the HttpResponse object, or it should raise an exception.

Now let's compare both of the views and see the pros and cons of both of them.

1. Function-Based Views

- Function-based views are good for beginners. It is very easy to understand in comparison to class-based views.
- Initially when you want to focus on core fundamentals, using the function-based views gives the advantage to understand it. Let's discuss some pros and cons of it.

Pros:

1. Easy to read, understand and implement.
2. Explicit code flow
3. Straightforward usage of decorators.
4. Good for the specialized functionality.

Cons:

1. Code redundancy and hard to extend
2. Conditional branching will be used to handle HTTP methods.

Note: As we have discussed function-based views are easy to understand but due to the code redundancy in a large Django project, you will find similar kinds of functions in the views. You will find a similar kind of code is repeated unnecessarily.

- All the above cons of FBVs you won't find in class-based views. You won't have to write the same code over and over in your boilerplate.

2. Class-Based Views:

- Class-based views are the alternatives of function-based views. It is implemented in the projects as Python objects instead of functions.

- Class-based views don't replace function-based views, but they do have certain advantages over function-based views. Class-based views take care of basic functionalities such as deleting an item or add an item.
- Using the class-based view is not easy if you're a beginner. You will have to go through the documentation, and you will have to study it properly.
- Once you understand the function-based view in Django and your concepts are clear, you can move to the class-based views.

Let's discuss the class-based views in detail.

Pros:

1. The most significant advantage of the class-based view is inheritance. In the class-based view, you can inherit another class, and it can be modified for the different use cases.
2. It helps you in following the DRY principle. You won't have to write the same code over and over in your boilerplate. Code reusability is possible in class-based views.
3. You can extend class-based views, and you can add more functionalities using Mixins.
4. Another advantage of using a class-based view is code structuring. In class-based views, you can use different class instance methods (instead of conditional branching statements inside function-based views) to generate different HTTP requests.
5. Built-in generic class-based views.

Cons:

1. Complex to implement and harder to read
2. Implicit code flow.
3. Extra import or method override required in view decorators.

3. Django Generic Class-Based View

- Creating a new object, form handling, list views, pagination, archive views all these things are the common use cases in a Web application. It comes in Django core, you can implement them from the module **django.views.generic.**
- Generic class-based views are a great choice to perform all these tasks. It speeds up the development process.
- Django provides a set of **views**, **mixins**, and **generic class-based views**. Taking the advantage of it you can solve the most common tasks in web development.
- The main goal is not to reduce the boilerplate. It saves you from writing the same code again and again. Modify **MyCreateView** to inherit from **django.views.generic.CreateView**.

For example:

```
from django.views.generic import CreateView

class MyCreateView(CreateView):
    model = MyModel
    form_class = MyForm
```

- You might be thinking that where all the code disappears. The answer is that it's all in **django.views.generic.CreateView.**

- You get a lot of functionality and shortcuts when you inherit from CreateView. You also buy into a sort of convention over configuration.' style arrangement. Let's discuss few more details…
- By default template should reside in /<modelname>/<modelname>_form.html. You can change it by setting the class attribute template_name and template_name_suffix.
- We also need to declare the model and **form_class** attributes. Methods you inherit from **CreateView** rely on them.
- You will have to declare **success_url** as a class attribute on the view or you will have to specify **get_absolute_url()** in the model. This is important for the view in your boilerplate else the view won't know where to redirect to following a successful form submission.
- Define the fields in your form or specify the fields class attribute on the view. Here in this example, you can choose to do the latter.

Look at the example given below to check how it will look like.

```
from django import forms

from . models import MyModel
class MyModelForm(forms.ModelForm):
  class Meta:
    model = MyModel
    fields = ['name', 'description']
```

Conclusion:

- It is still a debate among developers that which one is good to use. Class-based views or function-based views? We have discussed the pros and cons for both of them but it totally depends on the context and the needs.
- We have mentioned that class-based views don't replace function-based views. In some cases, function-based views are better and in some cases, class-based views are better.
- In the implementation of the list view, you can get it working by subclassing the ListView and overriding the attributes. In a scenario where you need to perform the more complex operation, handling multiple forms at once, a function-based view will be a better choice for you.

Class Based Views Concept:

- Class Based view provide an alternative way to implement views as python objects insted of using Function based views.
- They do not replace the FBVs, But have certain differences and advantages when compared to function based views.
- In CBVs we can inherit the one class from another class.
- A View is callable which takes request and gives response.
- Django provides some classes which can be used as views.
- These classes allow us to structure our views and reuse code by implementing inheritance between the classes.
- Django is providing generic module which contains several predefined view classes.

  For example: django.views import generic
  from **django.views.generic import ListView, CreateView, DetailView, UpdateView, DeleteView**

ListView:

- We can use ListView class to list out or getting all records from model table(database).
- It is alternative way to ModelClassName.objects.all() in Function Based Views.

HOW to create template file for ListView:

- Django will identify template name automatically and we are not required to configure anywhere.
- But Django will always search for template file with the name as **modelName_list.html.**
- **Syntax : modelName_list.**html
  For example : contact_list.html if modelName is Contact

- Django will always search for template file in the following location.

  Syntaxt : templates/appName/modelName_list.html
  For example : templates/appName/contact_list.html

NOTE: Default CONTEXT OBJECT NAMEs

- By default django will provide context object to the template file with some default values.
- If we want to overide those default values then use **context_object_name** and **template_name** attributes in our class.

ListView

- context_object_name = modelName_list
- template_name = contact_list

DetailView

- template_name = modelname_detail.html
- context_object_name = modelName or object

CreateView and **UpdateView**

- template_name = modelname_form.html
- context_object_name = form

DeleteView

- template_name = modelname_confirm_delete.html
- context_object_name = object or modelName

Note : In urls.py file, create required urls to execute the both Function Based views and Class Based Views.

- Function Based Views are calling like **views.ViewName**
- Class Based Views are calling like **views.ViewName.as_view()**

Note : If we forgetting the **as_view()** as a suffix of our view name then django consider this name created by def keyword as Function Based View. But not treat like created by class keyword as Class Based View.
So finally we are getting errors.

success_url : It represents the target page url pattern which should be displayed by after successful operation like creating or deleting.

reverse_lazy() : This function will wait until deleting the record. next assigning the input url name to **success_url** .

```
from django.urls import reverse_lazy
success_url  = reverse_lazy('contact_list')
```

Open models.py

```
from django.urls import reverse

class Contact(models.Model):
  ----
  ----
  def get_absolute_url(self):
     return reverse('contact_list')
```

Create a project to perform CURD operations on database using Class Based View.

Step1: projectName : CBV_CURD_Project

step2: appName : students
Step3: Database Name : cbv_curd_db
Step4: Open settings.py file,

1. Add our appName inside INSTALLED_APPS section
2. Configure the Template path in TEMPLATE section.
3. Configure database details inside DATABASE section

Step5: Open models.py and create required models
Step6: Open project level __init__.py and write requeird code
Step7: Open the views.py and create Class based CURD operations views
Step8: Open urls.py and create requeird urls for mapping views
Step9: Create requeired tempaltes for views purpose.
Step10: Execute makemigrations, migrate, createsuperuser and runserver commands

Questions:

Q1. Why required application level urls in a Project ?

Q2. How to Mapping the Project level url to application level url ?