**Week 1-2**

First of all, in order to work with Django we have to install it and import it to our folder. From the below screenshot we see that we didn’t have Django, but then we installed it by the following command:

**pip install virtualenvwrapper-win**

Text

Description automatically generated

Then we are installing Django (version 3.2.9):

Text

Description automatically generated

Creating a new folder ‘projects’ where we will have all Django files:

Text

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

Finally, we are running local server at the following address: **http://127.0.0.1:8000/**

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Then we are creating Django template:

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [

'/home/html/amenu.com',

'/home/html/default',

],

},

{

'BACKEND': 'django.template.backends.jinja2.Jinja2',

'DIRS': [

'/home/html/jinja2',

],

},

]

**from** **django.template.loader** **import** render\_to\_string

rendered = render\_to\_string('my\_template.html', {'foo': 'bar'})

**from** **django.template** **import** engines

django\_engine = engines['django']

template = django\_engine.from\_string("Hello {{ name }}!")

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 developers to build a full-fledged web application in Django. If you’re an experienced Django developer then surely you might have been aware of the flow of the project. How things run in the boilerplate of Django and how data gets rendered to the user.

Django works on the MVT concept we mainly work on two types of views in it… 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)

First of all, 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.

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

**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:**

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

**Cons:**

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

**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:**

* 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.
* 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.
* You can extend class-based views, and you can add more functionalities using Mixins.
* 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.
* Built-in generic class-based views.

**Cons:**

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

**Code for our admin panel:**

from django.contrib.admin import AdminSite

from django.http import HttpResponse

from django.urls import path

from .models import Question

from django.contrib.auth.models import Group, User

from django.contrib.auth.admin import GroupAdmin, UserAdmin

class CustomAdminView(AdminSite):

def get\_urls(self):

urls = super().get\_urls()

my\_urls = [

path(r'my\_view/', self.admin\_view(self.my\_view))

]

urls = my\_urls + urls

return urls

def my\_view(self, request):

return HttpResponse("Hello, world.")

custom\_admin = CustomAdminView()

custom\_admin.register(Question)

custom\_admin.register(Group, GroupAdmin)

custom\_admin.register(User, UserAdmin)