**Django Templating system - How to make use of it efficiently**

**Introduction:**

The django template system is available to users (programmers) using django framework for creating web applications to allow swift creation of dynamic web pages with data from django models. The template system serves the “view” functionality of the MVC pattern implemented by django. Here, we will take a look at some of the features available with the django template mechanism. The templating system is actually quite extensive, and a full analysis of it is outside the scope of this document. I would suggest the reader to refer to the documentation for template system in django after reading this document. The documentation is available at “<https://docs.djangoproject.com/en/2.2/topics/templates/>”.

**The Templating System in General:**

Django provides the programmer with an excellent template system to serve the view component of MVC. The template system is quite flexible, and if the programmer wants to use template engine other than the default template engine, django allows that by the way of setting the “TEMPLATES” configuration parameter in the django settings file (settings.py). The TEMPLATES parameter may be set as follows:

*TEMPLATES = [*

*{*

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

*'DIRS': [],*

*'APP\_DIRS': True,*

*'OPTIONS': {*

*# ... some options here ...*

*},*

*},*

*]*

*#listing 1*

Django supports the Jinja template system apart from the default template system, and this may be used by setting the 'BACKEND' key with the value “django.template.backends.jinja2.Jinja2” in the TEMPLATES parameter.

Here, we will look at the default template system only, but the Jinja template system may be used in a very similar way from your python views.py functions that serve as controllers in the MVC pattern. (I will cover views.py in a later article, but for now the knowledge about views.py from my earlier article should be enough).

The django.template.loader module defines 2 functions to load a template from a views.py controller function: get\_template(template\_name, using=None) and select\_template(template\_name\_list, using=None). We will consider these 2 functions here.

**Using the Template System from Controller Code:**

The template system depends on the values computed in the views.py controller functions. The flow of logic for using templates is like this:

Step #1) You write a views.py function (taking a HttpRequest object as a parameter), and compute some values that you want to use in the template. These values may come from the database, and you may access the DB data using objects of your models.py classes. Each object of a models.py class is a record in the corresponding table. (More on this later).

Step #2) You store all these values in a dictionary object. You specify a certain key for it and place the value for the key as the value you computed or queried from the DB.

Step #3) You create a “django.template.Context” object from the dict created and populated in step #2.

Step #4) You call the “get\_template” function with the path to the Django template file as an argument. The template files normally have a .html extension as template files are HTML files, optionally with django template language code embedded in them (unless you are writing a very simple app, you will embed django template language code in that HTML). The return value of “get\_template” function is stored in a python variable.

Step #5) You “render” the HTML read from the template file in step #4 by calling the “render” method of the object returned by the “get\_template” call in step #4. The “render” method takes the context object created in step #3 as an argument.

Step #6) You return the return value of the “render” call mentioned in step #5 as a HttpResponse object. The views.py controller function ends here.

Step #7) Finally, you populate the template HTML file (referred to in step #4) using whatever HTML content you want, with keys of the python dictionary referred to in step #2 as variables in the HTML file. Each of these variables are enclosed between “{{“ and “}}” symbols. You may even put some logic into the HTML file using these variables in “if”, “for” or some other logical statement. The logic statements should be enclosed in “{%” and “%}” symbols.

**Example:**

Now that we have set down the flow of logic for creating and using templates, let us look at an example. I will assume that there is an application by the name “testapp”. Please refer to my earlier article (titled “How to set up a REST Service or a Web Application in Django” ) to see how to create a django app. The directory structure under “testapp” is like this:

*testapp/*

*\_\_init\_\_.py*

*admin.py*

*migrations.py*

*templates/*

*models.py*

*views.py*

There will be some other files in the above structure, but they are not relevant to django template system. So for the sake of simplicity, I have skipped them.

The “templates” directory above will contain the html file “testapp\_template.html”. The contents of the file are as follows:

*<!doctype html>*

*<html>*

*<head>*

*<title>Test App Template</title>*

*</head>*

*<body>*

*<center>*

*<p>{{ name }}</p>*

*<p>{{ birth\_place }}</p>*

*</center>*

*</body>*

*</html>*

*#listing 2*

Creating this file is specified as the step #7 above, in the list of steps to use django template system. However, it doesn't matter if we do this in the first place as it is not related to the steps from 1 through 6 above. Please note that we have used 2 variables, named “name” and “birth\_place” in the above HTML listing. We will need to define these variables as keys of the dictionary in our controller function that uses this HTML listing.

The corresponding controller function (in the file views.py) is as shown below:

*def test\_controller(request):*

*name = "Supriyo"*

*birthplace = "Kharagpur"*

*testdict = {}*

*testdict['name'] = name # note the key 'name' # step #2*

*testdict['birth\_place'] = birthplace # note the key 'birth\_place' # step #2*

*cxt = Context(testdict) # Creating context object - step #3*

*tmpl = get\_template("testapp\_template.html") # calling "get\_template" function - step #4.*

*testhtml = tmpl.render(cxt) # rendering html content - step #5.*

*return HttpResponse(testhtml) # returning rendered HTML as HttpResponse - step #6.*

*#listing 3*

Please note that the controller function above takes a HttpRequest object ('request') as parameter. The return value is a HttpResponse object.

“select\_template” is just like get\_template(), except it takes a list of template names. It tries each name in order and returns the first template that exists. In case it can't find any of the templates mentioned in the list, it raises the “TemplateDoesNotExist” exception.

**How to Make it Work:**

In order to make the above code work, create a django project and an application under the project using steps mentioned to create these in my earlier article titled “ How to set up a REST Service or a Web Application in Django”. Let's assume we have a project named “test” and an app named “testappp” under it. In the urls.py file, add the following code:

*urlpatterns = patterns('',*

*url(r'^/testapp/$', 'test.testapp.views.test\_controller ', name='test\_controller'),*

*)*

The above code basically means that if you hit the URL “<http://www.mydomainname.com/testapp/>”, then the controller function “test\_controller” will be invoked. The object path to test\_controller is “test.testapp.views.test\_controller” as it is defined in the views.py module. You would also need to add the following import statement at the top in the urls.py file:

*from django.conf.urls import patterns, include, url*

Next, you need to create the testapp\_template.html file in the “templates” directory. The file should contain the code specified in listing 2.

Next, add the following import statements in views.py file.

*from django.shortcuts import render*

*from django.http import HttpResponse*

*from django.template import Template, Context*

*from django.template.loader import get\_template*

Add the code in listing 3 in the views.py file under the above mentioned import statements.

Finally, start the django default web server, if it isn't already running. We will be using this webserver to run code that we are testing with. Later on, in one of my articles, I will show you how to use a production grade web server and application server to serve django app services. For now, please get into the project directory (named “test” as we have created a project named “test”). Run the following command in the directory:

*python manage.py runserver 0.0.0.0:8080*

Note that we are using the localhost and port 8080 to serve our page.

Point your browser “<http://localhost:8080/testapp/>”, and you should be able to see the webpage containing the values of the name and the birth\_place variables.

**Conclusion:**

We have just scratched the proverbial “tip of the iceberg” here. Django templates support inheritance, and they may also be “include”-ed in other template files. Template files may contain logic statements like “if”, “for”, etc. Templates are very handy to use, and they separate the view component from the controller logic in django. It is possible to implement the view component in the controller functions (in views.py) in django, but that is not advisable. By separating the view from the controller components, the separate files may be worked on simultaneously. For example, the web designer may be working on the view component (that is the django template file) while the programmer may be working on the controller component (the views.py file). This will save a lot of time as the components are separate.

As I have mentioned before, you should now go ahead and check out the template documentation. The link to it has been specified in the “Introduction” section. I wish you a great learning experience ahead.