Django Templates Concept

Overview

- The approach to implementing Django templates will be discussed in this article. The goal is to keep the Django application's UI better organized and to avoid redundant coding.
- Django has provided various mechanisms in the template engine to assist us in achieving this goal.
- In this tutorial, I'll show you how to use Django's built-in template tag block, extends, and include to make templates as easy to maintain as possible.

Introduction

- A Django template is essentially written in HTML, CSS, and Javascript in an HTML file.
- The Django framework handles and generates dynamic HTML web pages that are visible to the end user in an efficient manner.
- Django is primarily a backend framework, so we use templates to provide a front-end layout for our website.
- For developing web applications, a web framework uses a basic model view controller architecture software design pattern, but Django is a little different in a good way.
- It uses the Model-View-Template design pattern (MVT). MVT differs slightly from MVC. The main difference between the two patterns is that Django handles the Controller part (the software code that controls the interactions between the Model and the View), leaving us with the template.
- The template is made up of **HTML** and **Django Template Language (DTL).**

Django Templates

- Django requires an easy way to generate HTML dynamically. The most common method is to use Django templates.
- A template contains both the static and dynamic parts of the desired HTML output. Frontend developers have far more flexibility with templates than with MVC architecture.
- Different views can use the same template to display data in various formats. It saves all of the content rendered by the browser. This is the portion that the client sees.
- Django templates make use of a variety of other concepts, including template inheritance, tags, variables, filters, comments, and so on.

Why Django Template?

- Django templates adhere to the DRY (do not repeat yourself) design principle, which requires developers to refrain from repeating themselves while creating a Django application.
- Django templates are text documents or Python strings that have been marked up using the Django template language.

- Django separates logic and code from design. It is critical to understand that Django templates do not include any Python code embedded in HTML.
- The Django template engine is used to separate design from Python code, allowing us to create dynamic web pages.

Configuration

- To configure the Django template system, open the *settings.py* file and change the DIRS to the path to the templates folder.
- In general, the templates folder is created and kept in the Project directory, where ***manage.py*** is located.
- This templates folder contains all of the templates that you will make in various Django Apps for that project.

Steps to configure and Setup Django templates:

1. Inside your project directory, create a folder named templates.

```
├── db.sqlite3
├── project_name
├── manage.py
├── app_name
└── templates
```

2. Setup in Django settings where the templates are located, which helps locate the Django template file.

create TEMPLATE_DIR under the BASE_DIR

```
# Templates Directory

TEMPLATE_DIR = os.path.join(BASE_DIR,"templates")
```

This will return an absolute path to our project's templates directory, regardless of the operating system.

3. In **settings.py** scroll to the TEMPLATES.

```
TEMPLATES = [
    {
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [],
    'APP_DIRS': True,
    'OPTIONS': {
        'context_processors': [
            'django.template.context_processors.debug',
            'django.template.context_processors.request',
            'django.contrib.auth.context_processors.auth',
            'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Now add the newly created **TEMPLATE_DIRS** that you have created in the previous step in the DIRS of TEMPLATES(like this).

```
TEMPLATES = [
    {
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [TEMPLATE_DIR],
    'APP_DIRS': True,
    'OPTIONS': {
        'context_processors': [
            'django.template.context_processors.debug',
            'django.template.context_processors.request',
            'django.contrib.auth.context_processors.auth',
            'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Now save and close the **settings.py** file.

Using Django Templates

Compiling and Rendering Django Templates

Now, we will see how Django templates work.

1. Create an **index.html** inside the Folder template (as stated above).

```
├── DB.sqlite3
├── project_name
├── manage.py
├── app
└── templates
    └──index.html
```

You can also create a sub-folder in the template folder for different apps in the project.

2. **HTML code**

```
<!DOCTYPE html>
    <head>
        <meta charset="UTF-8">
        <title>Index</title>
    </head>
    <body>
        <h2>Django Templates</h2>
    </body>
</html>
```

3. To render the template, pass the HTML file in the **views.py** method(from the Django app) as stated below.

```
from django.shortcuts import render

# Create your views here.
def index(request):
    return render(request, "index.html")
```

Configuring Django to Load File Templates

- Add URL path in **urls.py** for the view function that contains the template file.

```
from django.contrib import admin
from django.urls import path
from app import views

urlpatterns = [
    path('/django_template', views.index, name='index'),
]
```

- Register the app inside the *INSTALLED_APPS* in **settings.py**, in case you forget to configure it during app implementation.

```
INSTALLED_APPS = [
    'app',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Run Server

Execute the following command in your IDE **terminal** to run the Django project and render the template you've created so far(check all the migrations are done).

```
python manage.py runserver
```

Django Template Language

- Django templates have their syntax for dealing with variables, tags, expressions, and so on. A context is used to retrieve the value at a web page when a template is rendered.
- Django includes a templating system known as Django template language (DTL). The template language of Django is intended to find a balance between power and simplicity. It is designed to feel familiar to those who are comfortable with HTML.

The main characteristics of Django Template language are:

Variables

- Variables return the value of the context, which is a dict-like object that maps keys to values. The context object sent by the view can be accessed in the template via **Django Template variables**.
- In simple words, whenever the template engine comes across a variable, it analyses and substitutes it with the result. Variable names can comprise any combination of alphanumeric characters and the underscore ("_"), but they cannot begin with an underscore and cannot be numbered.
- The dot (".") seems in variable sections as well, but it has a special meaning, as explained below. Importantly, variable names cannot contain spaces or punctuation.

Syntax

```
{{ variable_name }}
```

Example

```
<p>My Highest Qualifactions is {{ highest_qual }}.
       My favorite book is {{ fav_book }}. </p>

            <!-- With a context of {‘highest_qual’: ‘Bachelors in Computers’,
        ‘fav_book’: ‘The Alchemist’} --!>
```

Output

```
My Highest Qualification is bachelor in Computers.
My favorite book is The Alchemist.
```

Tags

- Tags allow for arbitrary logic to be used during the rendering process. A tag, for example, can output content, act as a control structure, such as an "if" statement or a "for" loop, retrieve data from a database, or indeed allow access to all other template tags.
- Tags are more complex than variables in that they generate text in the output, control flow with loops or logic, and load external data into the template that will be used by later variables.
- Some tags require starting and ending tags (for example, % tag%... tag contents... % endtag%).

Syntax

```
{% tag_name %}
```

Django includes about a dozen template tags by default. Some of the common ones are listed below:

- extends
- Comment
- cycle
- extends
- if
- for loop
- Boolean Operators
- include

Example

```
{% if age>10 %}
      Hello, {{ student.username }}.
{% endif %}
```

Filters

Django Template Engine includes filters for transforming variable and tag argument values. We've already covered the most important Django Template Tags. Tags cannot change the value of a variable, whereas filters can increase the value of a variable or modify it to suit one's needs. Filters can be used to modify variables for display. Filters can be "chained," which means that the output of one filter is implemented to the output of the next. Some filters accept arguments. Filter arguments containing spaces must be quoted. Django templates include approximately sixty built-in template filters.

Syntax

```
{{ variable_name | filter_name }}
```

Example

```
{{ value|filesizeformat }}
 <!-- Value is 123456789 --!>
```

Output

```
117.7 MB
```

Comments

Everything written in the comment syntax is ignored by the Django template engine. It is written as the reference point for the coders/developers. **Syntax**

- **For single line comment:**

```
{# comment added here #}
```

Example

```
{# framework used is #}
 Django
```

Output

```
Django
```

- **For multiple line comments:**

```
{% comment 'comment_name' %}
        comments you want to add
{% endcomment %}
```

Example

```
{% comment "Note to self" %}
        Re-check this block once again
{% endcomment %}
```

```
Django template
```

Output

```
Django template
```

Template Inheritance

- **Template inheritance** is the most effective and thus most complex feature of Django's template engine. Template inheritance allows you to create a base structure template that contains all of your site's common elements and describes components that child templates can override.
- In Django templates, the extends tag is used to indicate template inheritance. The same code must be repeated numerous times so we can prevent it by inheriting templates and variables by using **extends** keyword.

Syntax

```
{% extends 'template_name.html' %}
```

Example

```
├── app
└── templates
        ├──index.html
        ├──base1.html
        └──base2.html
```

In *index.html*, the following code of lines should be added on the top of the index file:

```
{% extends "./base2.html" %}
{% extends "./base1.html" %}
```

Conclusion

Have fun creating templates and drawing inspiration from existing HTML templates. Just a word of caution: Make sure to correctly configure your Django template files and load them. You'll be creating web apps much faster than me in no time.

A quick recap of the points you learned today:

- A Django template is a Python script that has been marked up with the Django template language and use to work with frontend/UI.
- To configure the Django template system, we have to add template file paths to the settings.py file.
- To render, we need to create the required methods in views.py and add the template files name as context.
- Add path of that view method in urls.py to load the template.
- Django templates make use of a variety of other concepts, including template inheritance, tags, variables, filters, comments, and so on.