Templates

Welcome to a pivot lesson of Django! So far we have been using a string to send responses to client side. Now we will learn the real stuff. Templates are basically folders containing HTML files that can be connected to views and bonded to URLs.

First off, let's take a look at Django project hierarchy once more. (for this project!)

```
MyProject/
    MyProject/
        <files.py>
    app1/
        <files.py>
    app2/
        <files.py>
```

if you want to have a template for each app, you must make a `templates` folder inside the directory of that app. We will make templates for `app1` and `app2` here. Oh! you also need to add a folder with the same name as the app inside the templates folder. Then, you'll be able to add your HTML files.

```
/MyProject
    /MyProject
        <files.py>

    /app1
        /templates
            /app1
                index.html
        <files.py>

    /app2
        /templates
            /app2
                index.html
        <files.py>
```

Let's add some HTML content shall we? (to `/MyProject/app1/templates/app1/index.html`)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
```

```
    <title>App 1</title>
</head>
<body>
    <h1>Hello, Templates!</h1>
    <h2>this is app2</h2>
</body>
</html>
```

And as usual, let's make a view. But this time, we will use a new method called `render_to_string()` which get's the path to `index.html` file and renders it to a string! This method is provided by `django.template.loader`.

p.s. you remember that to make a view you need you alter `/MyProject/app1/views.py`!

```
from django.template.loader import render_to_string
from django.http import HttpResponse

def index(rquest):
    response = render_to_string("app1/index.html")
    return HttpResponse(response)
```

Finally, let's bind it to a URL and give it a test.

```
# /MyProject/app1/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index)
]

# /MyProject/MyProject/urls.py
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('app1/', include("app1.urls"))
]
```

settings.py

We have already mentioned that whenever a new app is made, it must be added to `INSTALLED_APPS` list in `settings.py`. But we never really said why!?

Well if you remove that now, you'll see a couple of errors mentioning that Templates canot be found. The best-practice here is basically adding the app to `settings.py` and ask no more question (for now!). There is however, another way of introducing Templates to Django without adding the app in project setitngs.

Let's take a look at `TEMPLATES` in project settings.

```
from .MyProject.MyProject.settings import *

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',
            ],
        },
    },
]
```

In this python `list[JSON]` called `TEMPLATES` you can add the directory to your app templade (e. g. `BASE_DIR/"app1"/"templates"`) at the `DIRS` key.

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR/"app1"/"templates"], # BASE_DIR is defined
in settings.py
        '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',
            ],
        },
    },
]
```

Why add to INSTALLED_APPS?

With all that said, it's not so clear why we added the app to INSTALLED_APPS earlier. Well for starters, it's way easier to add a single word to a list instead of an entire path to a JOSN. But that's really not the reason.

By looking at the key `APP_DIRS` in the TEMPLATES we'll see it is set to `True`. That means that Django is initially configured to look for templated inside anything that is considered an `APP`, and anything that is added to INSTALLED_APPS is indeed and `APP`!

Nevertheless, there is more to INSTALLED_APPS than we mentioned, we'll learn about those in Database lessons later on.

Render Method

At the beginning of our journey, we witnessed a `render()` method imported to `views.py` file from `django.shortcuts`. Which we mentioned (or maybe not) that this is way cooler to use than `HttpResponse`.

What is does, is very basic. If you take a look at the behind-the-scene codes for `render()`, you'll see this.

```
from django.template import loader

def render(
    request, template_name, context=None, content_type=None,
status=None, using=None
):
    """
    Return an HttpResponse whose content is filled with the result of
calling
    django.template.loader.render_to_string() with the passed
arguments.
    """
    content = loader.render_to_string(template_name, context, request,
using=using)
    return HttpResponse(content, content_type, status)
```

You can that `render()` isn't doing anything so magical, yet gives us exactly what we need!

To use it ourselves we only need to return something like `render(request, 'path/to/file.html')` and Voalla!

```
#/MyProject/app1/views.py
def r_index(request):
    return render(request, "app1/index.html")
```

One significant point is that what we're rendering and sending back to client side is infact static! Meaning that we can't change the data in that HTML file based on the client side's request. However, if we take another look at the render behind-the-scene codes, we notice some parameter called `context`, good news, that solves our problem!

`context` is essentially a dictionary that can be rendered alongside our HTML file.

```
#/MyProject/app1/views.py
def r_index(request):
    data: dict = {
        'name':'Erfan',
        'family':'Rajati'
    }

    return render(request, "app1/render_index.html", context=data)
```

Now, how to use the data in that dictionary in our HTML file? By using the `{{ python_syntax }}` notation inside any tag you desire.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>App 1</title>
</head>
<body>
    <h1>Hello, Templates!</h1>
    <h2>this page is using render method.</h2>

    <h3>
        name: {{ name }}
    </h3>
    <h3>
        lastname: {{ family }}
    </h3>
</body>
</html>
```

This special syntax is called **DTL, Django Template Language.** Cool thing is that you can use a lot of Python Syntax in between two curly braces, There is things beyond Python syntax too! DTL is by itself an entire language based on Python.

Django Template Filters (By DTL)

There is a lot of features made available for django developers which is called Django Template Filters and Django Template Tags. Below, you'll find some examples of how you can use these filters in your HTML file.

```
{{ value|add:'2' }}
{{ first|add:second }}
```

If you're interested, visit this link to see more.

With that said, it's not like the end of the world if you don't learn DTF. You can handle all modifications in `.py` files on your own. At the next chapter, we'll take a look at Django Template Tags, which are, unlike filters, quite important to know!