Sunday, 27 November 2022

Django Notes

Filters

* You cannot use python keywords in your html template. But instead you can use filters.
* Use filters inside the the string interpolation in the template by adding a pipe symbol with the filter name
* Example title filter

{{ value | title }} - use this instead of capitalize() method

Tags and List

* Tags are another feature which is built into the DTL(Django Templating Language)
* There are a lot of tags
* Example for tag

It let’s you use to loop through the content

{% for month in months %}

{% endfor %} - end the for tag/block

* The reverse() function which redirects the request in the views does not work in templates. So we use url tag .

eg:- <a href = “{% url ‘monthly\_challenge’ month%}”

Here the monthly\_challenge is the name of the url which we defined in urls.py of the particular application and it should be mentioned in quotes. Quotes can be single or double quotes. Unlike reverse method we cannot use comma, so here we use white spaces to send arguments. Here the month is the argument. We can pass more than one arguments. We can also define names to the argument with =.

* url tag has no ending tag
* The DTL Tags are not visible to the client. Everything that is part of DTL is evaluated in the server.
* **If Tag**
* The if tag is used to add a condition in the template. It is used to render contents conditionally
* There is if tag, elif tag, else tag. We should end the if condition with endif tag
* We can use operators that are in python with if tag.
* Example:-

{%if text is not None%}

<h2>{{text}}</h2>

{%else%}

<p>There is no challenge</p>

{%endif%}

**-Block Tag**

* It is another django template tag which lets the content to be injected later.

eg :- {% block page\_title%}My challenges{%endblock%}

The page\_title is the name of the block. The end of the block is specified by the {%endblock%} tag. The content that is written between the opening of the block and ending of the block which is not mandatory but is a fallback content when the block is not present.

-You can have more than one block. It can be anywhere in the html page.

**-Template Inheritance**

* In the html template most of the things are common for every template such as header, navigation bar, etc. The idea behind template inheritance is that the core structure of template is maintained in a single template and based on condition content is dynamically injected.
* The core structure is stored in a folder that is of the same level of app folders
* To extent the template from the base template we can use the extends tag in the child template. This must be the first tag of the child template.
* Eg:-

{%extends “../../../templates/base.html”%}

here the two ‘.’ And a ‘/‘ is pointing one directory level up. Here the child template is residing in challenges/templates/challenges/index.html. so by giving like the above we can move to the root level folder of our project where templates/base.html is residing which is the parent.

* The above method is cumbersome because we have to give the relative path of the parent template. So another way is to provide a relative path.
* Eg :- {%extends “base.html”%}

and defining it in the settings.py of the project under Templates , DIRS

Like BASE\_DIR/ “templates”

* In the child template we specify the content like shown below.

eg:- {%block page\_title%}

All Challenges

{%endblock%}

{%block content%}

<ul>

<li> </li>

</ul>

{%endblock%}

Note that the name of the blocks should be the same as that we defined in the parent template.

We don’t need to provide the base tags of html 5 for the child template.

* There can be more than one parent template/ base template.

**Partial Templates**

* Example of partial template is a navigation bar.
* The entire project may not need the same template, like in cases where the navigation bar should not be shown in particular pages.
* As an example we can create a folder under inside the app/templates/challenges named include and create an html file under it which have the header
* To use this we use a special flange tag {%include%}.
* Eg:- {%include “challenges/include/header.html”%}

Here the relative path to the partial template should be given in “”, and we can use it wherever we want.

* Note that the include tag has no ending tag.
* Also the included templates will have access to variables which are being used in the current template where the template is added.
* We can also pass arguments to the particular template which are only accessible by that template and not the template to which the template is included. We can do this by

{%include “challenges/include/header.html ” with active\_page=“index”%}

we can output the active\_page variable using

{{active\_page}} which will give the value index

* include tag must be in a block.

**404 Templates**

* The error responses should be shown in a template.
* For this create a template for 404 errors in the templates folder of the project
* We can extend other base templates in this template.
* As an example we can create a 404.html inside the templates folder of the project. Inside this we can extend the base.html which is the base template for all the templates in the project. Inside this we can add a block which specifies the error response

{%extends “base.html”%}

{%block page\_title%}

something went wrong

{%endblock%}

{%block content%}

Sorry we couldn’t find the template.

{%endblock%}

* To render the page we should use render\_to\_string method in views.py and pass the rendered page to the HttpErrorResponse() which is returned. Here we must not use render method because it returns a success response, and we don’t want a success response to handle an error case.

the render\_to\_string method should be imported as

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

* There is alternative to this approach which is Http404 class in django.http
* It is a class which we raise as an error.

ie, raise Http404()

* If we want we can pass any string as an argument to this Http404() method. But, by default this Http404() will look for a 404.html file in your template. So, in-order for this to work we need to place the template in root folder of the project.
* Also we need to set DEBUG = False in the settings.py file otherwise it will show the 404 page with information which is useful for debugging.
* **The second method should be only used for deployment. Otherwise the development server won’t work if DEBUG is set to False.**

**Static Files**

- Static files include all the css, images, js files.

* As a convention static files should be placed in static folder of the app. Inside the static folder create another folder whose name is same as the name of the app.
* As an example inside the challenges app

Create static/challenges.

* To ensure that the static files are accessible by django we must ensure that in settings.py of the project under installed apps

‘django.contrib.staticfiles’ are present.

* Also ensure STATIC\_URL is present in the settings.py, this is further required for deployment purposes.
* By default django will look for folder named static and automatically detect those and add those.
* **The {%load static%} tag is used to load static files.**
* Incase of css files after you use the above keyword use <link rel=“stylesheet” href=“path to css file”>
* Ensure that the <link> tag is added in a block
* Since we loaded static using the tag, in the link tag’s href attribute we can use {%static “path”%} to load the css file.
* This static is similar to url tag we used
* Example :- {%static “challenges/challenges.css”%}
* In-order to apply css changes we need to restart the development server.

**Adding Global Statics Files**

* The above method is used to add static files or styles locally to an app. If you want to apply them globally we need to create a folder called static in the root of the project folder, similar to which we created templates folder.
* Inside the statics folder create a file called styles.css in which you can create global styles.
* You can add this global styling the the base template similar to which we added local static css files. The only exception is that here the <link> tag need not be in a block since there are html <tags>. We use the same {%load static%} tag here also and specify the url with {%static “style.css”%}.
* There is one problem in the above step. That is django searches for static files in the static folder. This is only true in case of apps. In case of projects we need to add a special django settings to settings.py file.
* Like we did for global templates,

add **STATICFILES\_DIRS = [**

**BASEDIR/ “static”**

**]**

**Project**

* To create a new project use the command

django-admin startproject project\_name

eg:- django-admin startproject my\_site

-To create a new app inside your project navigate to the root folder of your project and use the command

python3 manage.py startapp app\_name

eg:- python3 manage.py startapp blog

* to run the server use python3 manage.py run server
* First create a urls.py file for the app. Inside this add urlpattern[] list. Inside this list add path() function which is imported from django.urls
* To create an index use empty quotes in the path””. Otherwise if you need to set a particular name for the path we can use it.
* If you want a dynamic url we use <> in the path

eg:- path(“posts/<slug>”). The value passed after posts/ will be passed to the corresponding view which we define. We can define any name inside <>. Most commonly used name is slug.

We have seen path transformers (<str:month> , <int:month>) which triggers a path if the value passed is in string format or integer format.

Similarly we have a slug type. It ensures that the values passed in slug format can trigger the url.

Eg: /my-first-post is in slug format. That is it will only have strings, numbers and hyphens.

We define this transformer in the path function as <slug:slug>

* Then we define the views for the app in views.py of the app. First create a function for the index page which will have request as a parameter which is passed automatically by django.
* Then import the views of the current app to the urls by using from . Import views.
* Then add the views to the path() like views.function\_name
* Optionally we can give names to the paths with name=“name”
* The next step is the to define the urls of the path in the urls.py file of the project. To do that add a new path() function to the urlpatterns[] list and inside the path function define a name for the leading path segment for the app. Then as the second argument to the path() function use include() to import the urls file of the project like include(“app\_name.urls”)

it would look like path(“app\_name/“,include(“app\_name.urls”)

The “app\_name” is optional since we can also use “”. If you use “” we will get the website if we type localhost:8000/. Otherwise we will have to give localhost:8000/app\_name/ to see the starting page.

Remember to import the include() function from django.urls

* The next step is to add templates. For that we need to create a templates folder in the app folder for the app specific template. As a best practice create a folder inside the templates with the same name of the app. Ie, templates/app\_name. Inside this we can add the html files.
* If you need some global templates from which your app can inherit, we can create templates folder in the root of the project. Inside this you can create html files to include the skeleton code for the app. In the skeleton code define blocks that will be replaced by the contents of the children templates.

**Note : Make sure you place the global templates folder in the same level as the manage.py file. Otherwise we will get template not found exception**

* In order for the global templates to be accessible by django we either have to give the full path of the folder of the global template in the child template to which we are inheriting the base template. The much cleaner approach is to define them in settings.py of the project inside the templates[] list under DIRS. There inside the list give BASE\_DIR/“templates” to select the global template.
* Now we can inherit the base template to the the desired child templates using extends tag of dtl.

eg:- {% extends ‘base.html’%}

* One important thing we need to do is that we have to specify the app inside the INSTALLED\_APPS[] list of the settings.py file of the project. If we don’t do this the templates folder of the app will not be considered.
* After this we need to make sure that when the user visits the home page we need to display the template. For this we need to modify the views.py file of the app. To make django render a page we need to import render from django.shortcuts. It might be automatically imported. In the body of the corresponding view function return render() method with arguments request and path of the template

eg:- return render(request,”blog/index.html”)

here we need to pass the subfolder name of the templates folder of the app which is blog.

* The next step is to add content to the html page.
* Then create a folder named static where we store the css files globally for the project. This should be in the same level as manage.py file of the project. Inside this folder create a css file called app.css which is a common generic file.
* Then create a folder your app specific css files. For this inside the project create a folder named static and inside the static folder create a folder which has the same name of the app.

ie, static/blog

* After defining the styles in the respective files, define the link tag in the base.html file. For that first we need to import static like:

{% load static %}

Then we need to use the static dtl keyword to reference the correct css file.

ie, <link rel=“stylesheet” href=“{%static “app.css”%}”>

But in-order for this to work we need to tell django that there is a global static folder.

For this go to the settings.py file of the project and create a list called STATICFILES\_DIRS[]. Inside this define BASE\_DIR/“static”

Ie,

STATICFILES\_DIRS = [

BASE\_DIR / “static”

]

* The next step is add app level css styles. For this import static first in the template of the app. Then mention the link tag inside the block which is defined for adding css styles. Here use the static to point to the app specific css file

Ie, {%block css\_files%}

<link rel=“stylesheet” href=“{%static “blog/index.css”%}”/>

{%endblock%}

[All posts not visible check?]