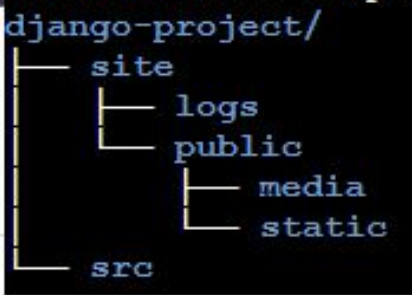
Django

site structure:



django-admin startproject projectname

python manage.py start app

settings.py = add app name to

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'projectname',

]

python manage.py runserver

view project on

<http://127.0.0.1:8000/>

create first view by function base view:

from django.http import HttpResponse

def home (request):

return HttpResponse("this is home page")

and set urls :

from django.urls import path

from . import views

urlpatterns = [

path('home' , views.home),

]

use the name for one url

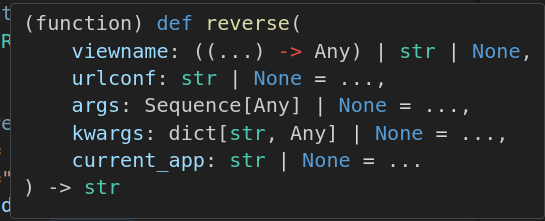
urlpatterns = [

path('home' , views.home , name='home'

),

]

and use reverse function to use url name instead of static url



url\_path = reverse('a',args=[day])

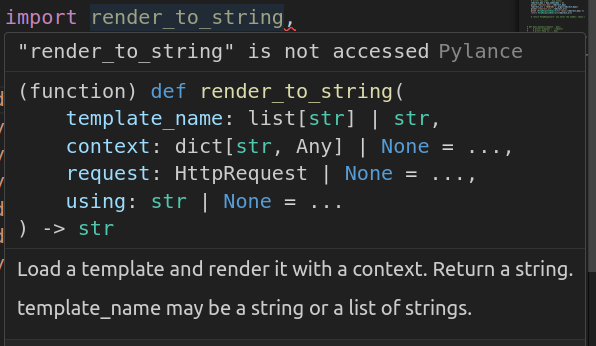
to redirect one url to another url

from django.http import , HttpResponseRedirect

HttpResponseRedirect(redirect\_url)

use render to string for serve one template and context to html

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



use render function for render template and context

from django.shortcuts import render

def days\_dynamic(request, day):

day\_list = list(days.keys())

context = {

'days':day\_list

}

return render(request , 'challenges/index.html' , context)

good to reed :

The Django template language: for Python programmers

https://docs.djangoproject.com/en/5.0/ref/templates/api/

DTL

The Django template language

https://docs.djangoproject.com/en/5.0/ref/templates/language/

It’s designed to feel comfortable to those used to working with HTML.

## Templates

A template is a text file. It can generate any text-based format (HTML, XML, CSV, etc.).

A template contains variables, which get replaced with values when the template is evaluated, and tags, which control the logic of the template.

## Variables

{{ variable }}

## Filters

{{ name | lower }}

## Tags

{% tag %}

{% **for** athlete **in** athlete\_list %}

<**li**>{{ athlete.name }}</**li**>

{% **endfor** %}

{% **if** athlete\_list|length > 1 %}

Team: {% **for** athlete **in** athlete\_list %} ... {% **endfor** %}

{% **else** %}

Athlete: {{ athlete\_list.0.name }}

{% **endif** %}

## Comments

{# #}

## Template inheritance

Inheritance from base.html

{% **extends** "base.html" %}

{% **block** title %}My amazing blog{% **endblock** %}

{% **block** content %}

{% **for** entry **in** blog\_entries %}

<**h2**>{{ entry.title }}</**h2**>

<**p**>{{ entry.body }}</**p**>

{% **endfor** %}

{% **endblock** %}

**url template tag**

{% **url** 'some-url-name' v1 v2 %}

<h1>{%for b in days %}</h1>

<h1><li> <a href="{% url 'a' b %}">{{b}}-- {{b|title}}</a></li></h1>

<h1>{%endfor%}</h1>

extend tag

{% extends "base.html" %}

include tag

{% include "challenge.html"%}

Http404 and HttpResponseNotFound

from django.http import Http404 , HttpResponseNotFound

raise Http404

Making queries

https://docs.djangoproject.com/en/5.0/topics/db/queries/

Once you’ve created your data models, Django automatically gives you a database-abstraction API that lets you create, retrieve, update and delete objects. This document explains how to use this API. Refer to the data model reference for full details of all the various model lookup options.

Models

https://docs.djangoproject.com/en/5.0/topics/db/models/

A model is the single, definitive source of information about your data. It contains the essential fields and behaviors of the data you’re storing. Generally, each model maps to a single database table.

The basics:

Each model is a Python class that subclasses django.db.models.Model.

Each attribute of the model represents a database field.

With all of this, Django gives you an automatically-generated database-access API; see Making queries.

**Slug field**

read about that on django document

**Models**

read about model and method

sample model code:

class Product(models.Model):

title = models.CharField(max\_length=300)

price = models.IntegerField()

rating = models.IntegerField(validators=[MinValueValidator(1), MaxValueValidator(5)] , default=0)

short\_description = models.CharField(max\_length=360 , null=True)

is\_active = models.BooleanField(default=False)

slug = models.SlugField(default="" , null=False , db\_index=True)

def get\_absolute\_url(self):

return reverse('product-detail' , args=[self.slug])

def save(self, \*args, \*\*kwargs):

self.slug = slugify(self.title)

return super().save(\*args, \*\*kwargs)

def \_\_str\_\_(self):

return f"{self.title} - {self.price}"

aggregate

study about aggregate in python

**order\_by**

products = Product.objects.all().order\_by("title") #assending

products = Product.objects.all().order\_by("-title") #desinding

order by

اردر کردن یا فیلتر کردن در کدهای پایتون نباید انجام شود و در دیتابیس و کوری زدن به دیتا بیس باید انجام گردد

**Add module to admin**

from django.contrib import admin

from . import models

admin.site.register(models.Product)

**Add property to admin**

class ProductAdmin(admin.ModelAdmin):

# readonly\_fields = ['slug']

prepopulated\_fields={

'slug':['title']

}

admin.site.register(models.Product , ProductAdmin)

read more about django admin on django document

**structure of Mdels Code**

from typing import Iterable

from django.db import models

from django.core.validators import MinValueValidator , MaxValueValidator

from django.urls import reverse

from django.utils.text import slugify

class ProductTag(models.Model):

tag = models.CharField(max\_length=300 , verbose\_name='تگ محصول')

class Meta:

verbose\_name = 'تگ محصول'

verbose\_name\_plural= 'تگ های محصولات'

def \_\_str\_\_(self):

return self.tag

class ProductCategory(models.Model):

title = models.CharField(max\_length=300 , verbose\_name='عنوان')

url\_title = models.CharField(max\_length=300 , verbose\_name='عنوان در url')

def \_\_str\_\_(self):

return self.title

class Meta:

verbose\_name ='دسته بندی'

verbose\_name\_plural = 'دسته بندی ها'

class ProductInformation(models.Model):

color = models.CharField(max\_length=200 , verbose\_name='رنگ')

size = models.CharField(max\_length=200 , verbose\_name='اندازه')

def \_\_str\_\_(self):

return f'({self.color} - {self.size} )'

class Meta:

verbose\_name ='اطلاعات تکمیلی محصول '

verbose\_name\_plural = 'اطلاعات تکمیلی محصولات'

class Product(models.Model):

title = models.CharField(max\_length=300)

product\_information = models.OneToOneField(

ProductInformation,

on\_delete=models.CASCADE,

related\_name='product\_information',

null=True,

blank=True)

category = models.ForeignKey(ProductCategory

, on\_delete=models.CASCADE ,

null=True ,

related\_name='products',

verbose\_name='دسته بندی')

product\_tags = models.ManyToManyField(ProductTag)

price = models.IntegerField()

rating = models.IntegerField(validators=[MinValueValidator(1), MaxValueValidator(5)] , default=0)

short\_description = models.CharField(max\_length=360 , null=True)

is\_active = models.BooleanField(default=False)

slug = models.SlugField(default="" , null=False , db\_index=True , blank=True)

def get\_absolute\_url(self):

return reverse('product-detail' , args=[self.slug])

def save(self, \*args, \*\*kwargs):

self.slug = slugify(self.title)

return super().save(\*args, \*\*kwargs)

def \_\_str\_\_(self):

return f"{self.title} - {self.price}"

class Meta:

verbose\_name ='محصول '

verbose\_name\_plural = 'محصولات '

**Django Render Partial**

use this app to load view in html page

https://pypi.org/project/django-render-partial/

pip install django\_render\_partial

Add 'django\_render\_partial' to INSTALLED\_APPS

Write a partial view:

def partial\_view(request

return render(request, 'partial\_view.html'

{% load render\_partial %}

{% render\_partial 'partial\_view'%}

**start working with FORM**

**work with form**

https://docs.djangoproject.com/en/5.0/topics/forms/

**widgets in form**

https://docs.djangoproject.com/en/5.0/ref/forms/widgets/

1-creat html form in html templates

tem[late.html

<**form** action="/your-name/" method="post">

<**label** for="your\_name">Your name: </**label**>

<**input** id="your\_name" type="text" name="your\_name" value="{{ current\_name }}">

<**input** type="submit" value="OK">

</**form**>

2- create form in form.py

forms.py[¶](https://docs.djangoproject.com/en/5.0/topics/forms/" \l "id3)

**from** **django** **import** forms

**class** **NameForm**(forms.Form):

your\_name = forms.CharField(label="Your name", max\_length=100)

views.py[¶](https://docs.djangoproject.com/en/5.0/topics/forms/" \l "id4)

**from** **django.http** **import** HttpResponseRedirect

**from** **django.shortcuts** **import** render

**from** **.forms** **import** NameForm

**def** get\_name(request):

*# if this is a POST request we need to process the form data*

**if** request.method == "POST":

*# create a form instance and populate it with data from the request:*

form = NameForm(request.POST)

*# check whether it's valid:*

**if** form.is\_valid():

*# process the data in form.cleaned\_data as required*

*# ...*

*# redirect to a new URL:*

**return** HttpResponseRedirect("/thanks/")

*# if a GET (or any other method) we'll create a blank form*

**else**:

form = NameForm()

**return** render(request, "name.html", {"form": form})

#### The template[¶](https://docs.djangoproject.com/en/5.0/topics/forms/" \l "the-template)

We don’t need to do much in our **name.html** template:

<**form** action="/your-name/" method="post">

{% **csrf\_token** %}

{{ form }}

<**input** type="submit" value="Submit">

</**form**>

انواع viewها

type of views

Model Form

class base views

TemplateViews

ListView

DetailView

CreatView > create form direct from model

**Creating forms from models¶**

https://docs.djangoproject.com/en/5.0/topics/forms/modelforms/

CSS styling in Django forms

class MyForm(forms.Form):

myfield = forms.CharField(widget=forms.TextInput(attrs={'class': 'myfieldclass'}))

or

class MyForm(forms.ModelForm):

class Meta:

model = MyModel

def \_\_init\_\_(self, \*args, \*\*kwargs):

super(MyForm, self).\_\_init\_\_(\*args, \*\*kwargs)

self.fields['myfield'].widget.attrs.update({'class': 'myfieldclass'})

or

class MyForm(forms.ModelForm):

class Meta:

model = MyModel

widgets = {

'myfield': forms.TextInput(attrs={'class': 'myfieldclass'}),

}

**Overriding the default fields**¶

https://docs.djangoproject.com/en/5.0/topics/forms/modelforms/