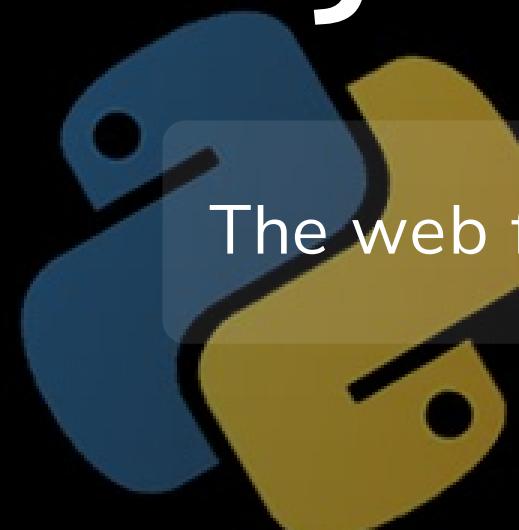


Django

The web framework for perfectionists with deadlines.



djang^o

Outline

1. Python intro
2. django Basics + models
3. Views, Forms, Templates
4. HTML, CSS, UI.
5. *Bonus #1 Deploying to internet*
6. *Bonus #2 API & react app*



Python basics

Speedrun Python basics



Basic operations (Arithmetic)

```
1 + 2 # Addition
```

```
1 - 2 # subtraction
```

```
2 * 3 # multiplication
```

```
9 / 2 # division
```

```
9 // 2 # floor division
```

```
2 ** 3 # exponent
```

```
5 % 2 # modulus
```

Basic operations (Comparison)

```
2 == 3 #Is it equal?
```

```
1 != 2 #Is it NOT equal?
```

```
7 < 11 #Is it lesser?
```

```
5 > 9 #Is it greater?
```

```
67 >= 88 #Is it greater than or equal?
```

```
24 <= 666 #Is it lesser than or equal?
```

Basic operations (Logical)

```
(1 < 2) or (3 < 5) #If either of the 2 operations returns true, the final result will also be true  
  
(4 = 3) and (10 ≠ 10) #Both operations need to return true in order for the final result to be true  
  
not (5*3 = 15) #Reversal
```

Variables

```
a = 10
```

```
print(a)
```

```
name = "minion"
```

```
print(name)
```

```
b = 42
```

```
B = "X-rays"
```

```
print(b,B)
```

```
D = 22
```

```
D = "Ant"
```

```
print(D)
```

Data types

Integer

```
int  
-1, 2, 0, 1000
```

Float

```
float  
0.1, 2.76, -9.45
```

Strings

```
str  
'python', 'NAXXATRA', "hello", "wonder"  
  
"""disco""",  
  
"""  
this is a multi-line string  
"""
```



Boolean

```
bool  
True, False
```

None

```
None
```

--

Check type

```
a = 10  
type(a)
```

Operations with variables

```
a, b = 10, 20 # assign multiple variables
```

try these operations with `a` and `b`

```
print(a + b)
```

```
print(a - b)
```

```
print(a * b)
```

```
print(a / b)
```

```
print(a < b)
```

```
print(a > b)
```

```
print(a == b)
```

```
del a,b #delete variables
```



Strings

```
# Strings are collection of characters
str1 = 'laka '
str2 = 'laka '
print(str1)
print(str2)
```

```
print(str1 + str2)
print(str1 * 5)
```

```
str1.upper(), str2.lower()
```

```
s1 = '1'
s2 = '2'
print(s1 + s2)
```

```
#Indexing (Collections in python are 0-indexed, i.e, the first element's index is 0)
print(str1[2])
```

```
#Slicing - str1[start : stop : step]
print(str1[1:5])
```



Type Conversion

```
a = 10  
b = str(a)  
c = float(a)
```

```
type(a)
```

```
type(b)
```

```
type(c)
```

```
d = int(b)
```

```
type(d)
```

```
a = input("Enter the first number: ")  
type(a)
```

```
a = int(input("Enter the first number: "))  
type(a)
```



Lists

Mutable collection of heterogenous items

```
# Mutable collection of heterogenous items
l = [1, 2, 'a', 0.4, False]
```

```
l[2] # accessing by index
```

```
l[2:4] # accessing a slice
```

```
l.append(42) # adding to end of a list
l
```

```
l.append([9,0]) # adding to a list
l
```



Tuples

Immutable collection of items

```
# Immutable collection of items  
t = (1, 2, 'a', 0.4, False)
```

```
t[2] # accessing by index
```

```
t[2:4] # accessing a slice
```

we cannot add, remove or modify items from a tuple



Dictionaries

Dictionaries are key value pairs

```
apple = {  
    "name": "Apple",  
    "price": 100,  
    "color": "red"  
}  
orange = {"name": "orange", "price": 50, "color": "orange"}
```



Dictionaries

Updating dictionaries

```
apple["price"] = 120 # updating a value
```

```
apple["flavour"] = "sweet" # adding a new key-value pair
```

```
apple.update({"calory":125,"type":"natural"}) # updating a dictionary with another dictionary
```

Conditional Statements

```
a, b, c = 10, 20, 30
```

```
if (a > b) and (a > c):
    print("a is the largest")
elif (b > a) and (b > c):
    print("b is the largest")
else:
    print("c is the largest")
```



Problem (FizzBuzz)

A practice problem to learn how to use conditional statements.

Fizzbuzz is a problem where you have to print the numbers from 1 to 100.

- But for multiples of 3 print "Fizz" instead of the number
- and for the multiples of 5 print "Buzz".
- For numbers which are multiples of both 3 and 5 print "FizzBuzz".

```
1 # example output
```

```
2
```

```
Fizz
```

```
4
```

```
Buzz
```

```
Fizz
```

```
7
```

```
8
```

```
Fizz
```

```
Buzz
```

```
11
```

```
Fizz
```

```
13
```

```
14
```

```
FizzBuzz
```

```
16
```



Sample Solution

```
for i in range(1, 101):
    if i % 3 == 0 and i % 5 == 0:
        print("FizzBuzz")
    elif i % 3 == 0:
        print("Fizz")
    elif i % 5 == 0:
        print("Buzz")
    else:
        print(i)
```

Challenge for you

Write the same program but using fewer conditional statements.



Solution 2

```
for i in range(1, 101):
    output = ""
    if i % 3 == 0:
        output += "Fizz"
    if i % 5 == 0:
        output += "Buzz"
    if output == "":
        output = str(i)
print(output)
```

```
for i in range(1,101):
    string = "".join("Fizz" if (i%3==0) else "")
    string = string + ("Buzz" if (i%5==0) else "")
print(i if(string=="") else string)
```

```
for number in range(1,101):
    if number % 3 == 0 and number % 5 == 0: result = "FizzBuzz"
    elif number % 3 == 0: result = "Fizz"
    elif number % 5 == 0: result = "Buzz"
    else: result=number
print(result)
```



Solution 3

```
for i in range(1, 101):
    if i % 3 == 0:
        if i % 5 == 0:
            print("FizzBuzz")
        else:
            print("Fizz")
    elif i % 5 == 0:
        print("Buzz")
    else:
        print(i)
```

```
for number in range(1,101):
    result = number
    if (number % 3 == 0): result = "Fizz"
    if number % 5 == 0:
        if type(result) == str:result += "Buzz"
        else: result = "Buzz"
    print(result)
```



Session 2

Django basics



Django

Django is a batteries-included web framework for Python.

- can create robust web applications
- easy to get started and learn
- uses an ORM (Object Relational Mapping)
- uses a Model-View-Template (MVT) pattern
- can be used to create a RESTful API (Representational State Transfer) (REST)
 - `(GET, POST, PUT, DELETE)`

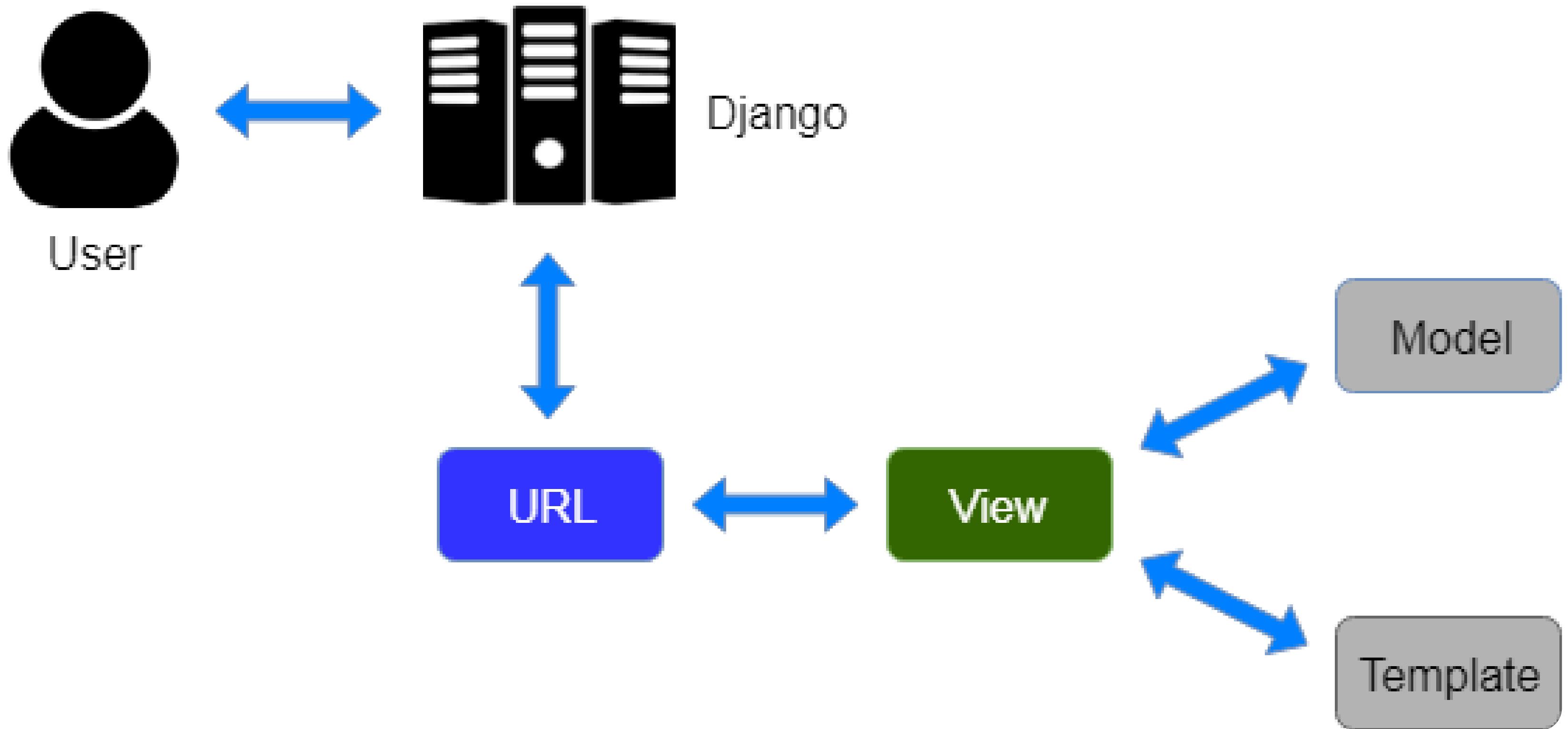


Django ORM

Django ORM is an application layer that allows you to interact with a database without writing any raw SQL.



MVT (model-view-template)



Classes & inheritance

```
class Human:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
        self.hands = 2  
        self.nose = 1  
        self.eyes = 2  
  
    def talk(self):  
        print("Hello")
```

```
def run(self):  
    print("I am running")
```

```
class Programmer(Human):  
    def __init__(self, name, age, language):  
        super().__init__(name, age)  
        self.language = language  
  
    def code(self):  
        print("I am coding")
```



def computer_skill(self):



MVT in code

models.py

```
class Post(models.Model):
    title = models.CharField(max_length=200)
    body = models.TextField()
    date = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title
```



Views

views.py

```
class PostView(View):
    """
    Views for post
    """

    def get(self, request):
        """url to get all posts"""
        posts = Post.objects.all()
        return render(request, 'blog/index.html', {'posts': posts})

    def post(self, request):
        """URL to create a post"""
        title = request.POST.get('title')
        body = request.POST.get('body')
        post = Post(title=title, body=body)
        post.save()
        return redirect('/')
```



Template

blog/index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <h1>Blog</h1>
    <ul>
      {% for post in posts %}
        <li>{{ post.title }}</li>
      {% endfor %}
    </ul>
    <h2>create a post</h2>
    <form action="/blog/create" method="post">
      <input type="text" name="title" placeholder="Title" />
      <textarea
        name="body"
        cols="30"
        rows="10"
        placeholder="Your post">
    </textarea>
```



layout: cover

Getting started with a new django app



Steps to start

Setup pipenv. Ignore this step if you have already setup pipenv/venv.

```
# install pip and venv  
python3 -m pip install wheel  
# upgrade pip  
python3 -m pip install -U pip  
  
# install pipenv  
python3 -m pip install pipenv  
# cd $projectroot/  
cd /to/path/of/the-project/  
  
# create virtual environment  
pipenv install  
# this might take a few minutes
```



Status checks

```
python -V  
  
pip -v  
  
# installing pipenv  
pip install pipenv --user  
  
# checking pipenv installation  
pipenv -h
```

possible errors:

```
pipenv shell 'pipenv' is not recognized as an internal or external command, operable program or batch file.
```



Windows troubleshooting

for pipenv users. Please ignore this step if you have already setup venv.

1. Press the `Windows key+X` to access the Power User Task Menu.
2. In the Power User Task Menu, select the `System` option.
3. In the About window, click the `Advanced system settings` link under `Related settings` on the far-right side.
4. In the System Properties window, click the Advanced tab, then click the `Environment Variables` button near the bottom of that tab.
5. In the `Environment Variables` window, highlight the `Path` variable in the System variables section and click the `Edit` button. Add or modify the path lines with the paths you want the computer to access. Each directory path is separated with a semicolon, as shown below.

Second, replace your `<username>` in the following paths and add them to the PATH environment variable:

```
c:\Users\<username>\AppData\Roaming\Python\Python38\Site-Packages  
C:\Users\<username>\AppData\Roaming\Python\Python38\Scripts
```



Getting started

windows instructions

```
# pip install django  
  
py -m pip install Django  
  
# check django version  
py -m django --version
```

change into your desired project directory and run the following commands:

```
django-admin startproject blog
```

Run the development server:

```
py manage.py runserver
```

press **Ctrl + C** to stop the server.



Start a new app

Apply database migrations. This step will create a new file `db.sqlite3` in your project root, if it didn't exist already.

```
py manage.py migrate
```

create a new app named posts where we can setup a model and views

```
py manage.py startapp posts
```

and create a superuser (admin), using which we can manage the project from the admin panel

```
py manage.py createsuperuser
```

this should bring up a prompt that looks like this:

```
Username (leave blank to use 'user'): admin
```

```
Email address: admin@gamil.com
```

```
Password:
```

```
Password (again):
```



Creating models

This file would live in your `project folder > posts > models.py`

```
from django.db import models
from django.utils.translation import gettext_lazy as _

class Posts(models.Model):
    """
    Model for the posts
    """

    title = models.CharField("Title of the post", max_length=200)
    content = models.TextField("Content of the post")
    created_at = models.DateTimeField("Created at", auto_now=True, editable=True)
    cover = models.ImageField(null=True, blank=True)

    class Meta:
        verbose_name = _("Post")
        verbose_name_plural = _("Posts")

    def __str__(self):
        return self.title

    def get_absolute_url(self):
```



Adding app to settings

`settings.py` controls the configuration of the project.

it contains a section called `INSTALLED_APPS` that contains a list of all the apps that are installed in the project.

open `[project_name] > settings.py` and add find the line below:

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



Installing necessary libraries

```
pip install Pillow
```

Make migrations

This will create necessary migrations for our new posts app.

```
python manage.py makemigrations
```

desired output:

```
Migrations for 'posts':  
  posts/migrations/0001_initial.py  
    - Create model Posts
```

this will apply the created migrations to database.

```
python manage.py migrate
```

desired output:

```
Operations to perform:  
  Apply all migrations: admin, auth, contenttypes, posts, sessions  
Running migrations:  
  Applying posts.0001_initial ... OK
```



Registering model in admin interface

Open up admin.py in posts app and add the following:

```
from django.contrib import admin

from posts.models import Posts

class PostsAdmin(admin.ModelAdmin):
    list_display = ('title', 'created_at')
    search_fields = ('title',)
    list_filter = ('created_at',)
    ordering = ('-created_at',)

admin.site.register(Posts, PostsAdmin)
```



Creating a new post

Open up the admin panel and create a new post. localhost:8000/admin/

Configuring additional settings

`settings.py` also contains sections for the `STATIC_URL` and `MEDIA_URL` variables, as well as `TEMPLATES` and `MIDDLEWARE`.

In the beginning of the file, add the following:

```
import os
```

let us first configure the `STATIC_URL` variable. we will use `STATIC_URL` to tell django where to find the static files.

```
STATIC_URL = "/static/"
```

configuring the `MEDIA_URL` variable. we will use `MEDIA_URL` to tell django where to find the media files (like images videos etc).

```
# Base url to serve media files  
MEDIA_URL = '/media/'  
  
# Path where media is stored  
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
```



Adding media url to settings

Open up `urls.py` and update it with the following:

```
from django.contrib import admin
from django.urls import path
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
]

if settings.DEBUG:
    urlpatterns += static(
        settings.MEDIA_URL,
        document_root=settings.MEDIA_ROOT
)
```



Setting up timezone

In `settings.py` we can set the `TIME_ZONE` and `LANGUAGE_CODE` variable.

```
LANGUAGE_CODE = 'en-in'  
  
TIME_ZONE = 'Asia/Kolkata'
```

this should give us local timezone, in all our interfaces



Setting up views for our blog-posts

in `posts/views.py` we can create a view for our blog-posts. these views will be used to render the blog-posts.

```
from django.views import generic
from posts.models import Posts

class PostListView(generic.ListView):
    """
    View to see all the posts in a list
    """
    model = Posts
    queryset = Posts.objects.all().order_by('-created_at')
    template_name = 'posts/index.html'

class PostDetailView(generic.DetailView):
    """
    View to see a single post in detail
    """
    model = Posts
    template_name = 'posts/detail.html'
```



Setting up urls for our blog-posts

in `posts/urls.py` we can create urls for our blog-posts.

```
from django.urls import path
from posts.views import PostListView, PostDetailView

urlpatterns = [
    path('', PostListView.as_view(), name='index'),
    path('<int:pk>/', PostDetailView.as_view(), name='detail'),
]
```



Setting up templates for our blog-posts

we need to first configure the `templates` directory, and add it to the `TEMPLATE_DIRS` variable.

1. Create a `templates` directory in the root directory of the project.
2. Add the `templates` directory to the `TEMPLATE_DIRS` variable in `settings.py`.

```
TEMPLATES_DIRS = os.path.join(BASE_DIR, 'templates')
```

3. Now in `settings.py` find the `TEMPLATES` variable and add the following:

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [TEMPLATES_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',  
            ],  
        },  
    },
```



Creating a basic template

We can define the basic layout of our application in `base.html` file.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <script src="https://cdn.tailwindcss.com"></script>
    <title>{{title}}</title>
  </head>
  <body>
    <h1>{{title}}</h1>
    {% block content %}
    
    {% endblock content %}
    
    <footer>
      <p>My new blog</p>
    </footer>
  </body>
</html>
```

