COMPLETE Django with ME

- Nishant Kolapkar

Django Framework

Dt. 25.12.2022

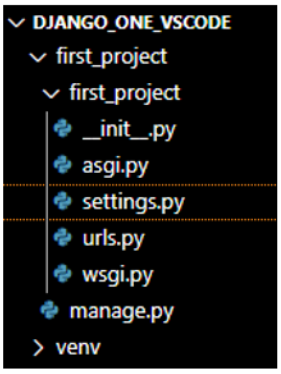
- Django is a free and open-source web framework. (1st created in 2003 for newspaper)

- **Virtual Environment or venv:**

- **Environment with conda:**
  - **conda create - -name myEnv Django**
  - **activate myEnv**
  - **deactivate myEnv**

- **Environment with CMD python:**
  - **Python -m venv path_of_venv**
  - **To activate venv >> env_folder_name\Scripts\Activate.ps1**
  - **To deactivate venv >> env_folder_name\Scripts\Deactivate**

- **Project Creation using Django:**
  - **django-admin startproject project_name**



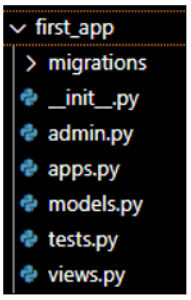
  - **_ _init_ _ .py**
    This is a blank Python script that due to its special name let's Python known that this directory can be treated as package.
  - **settings.py**
    This is where all settings are stored.
  - **urls.py**
    This python script will store all the URL patterns for project. Basically, the different pages of the web application.
  - **wsgi.py**
    This Python script act as the Web Server Gateway Interface. It will help in deploying to production.
  - **manage.py**
    Its associate with many commands to build web application.

  - **To run server:**
    python manage.py runserver (need project path if not work > cd project_path)
    http://127.0.0.1:8000/ (local host address)

  - **Django Application**
    - A Django Project is a collection of applications and configuration that when combined together will make up the full web application (i.e., complete website running with Django)
    - A Django Application is created to perform a particular functionality for your entire web application. For example, registration app, a polling app, comments app, etc.

- These Django Apps can then be plugged into other Django Projects, so it can be use with other apps
- To Create application:

  Python manage.py startapp app_name



- **migration folder**
  This directory stores database specific information as it relates to the models.
- **_ _init_ _ .py**
  This is a blank Python script that due to its special name let's Python known that this directory can be treated as package.
- **admin.py**
  To register models here which Django use them to admin interface with Django.
- **apps.py**
  Here can be place application specific configurations.
- **models.py**
  Here can be store application's data models.
- **test.py**
  Here can be store test functions to test your code.
- **views.py (Business Logic)**
  This is where you have functions that handle requests and return responses.

- **Step 1:**
  Register app in settings.py
  Search for INSTALLED_APP = [
  xyz (django default apps),
  'first_app',
  }

- **Step 2:**
  Code in views.py

first_project > first_app > views.py > ...

```
1   from django.shortcuts import render
2   from django.http import HttpResponse
3   def index(r):
4       return HttpResponse("Hello Web!")
5
```

- **Step 3:**
  General URL mapping views to urls.py

```
from django.contrib import admin
from django.urls import path
from first_app import views
urlpatterns = [
    path("admin/", admin.site.urls),
    path("", views.index, name="index"),
]
```

- **App Level URLs**

  - **include( ) function from django.urls app level URLs**: (need to check with recordings)
    The include( ) function allows us to look for a match with regular expressions and link back to our application's own **urls.py** file. (We will have to manually add in this **urls.py** file)

    - App level URLs is used for making urls.py file clean and modular.

    - **Step 1:**
      Copy and paste **urls.py** in every application folder
      Design code in **views.py**

    - **Step 2:**
      App level urls.py

```
from django.urls import path
from App_Two import views
urlpatterns = [
    path("apply/", views.apply),
]
```

    - **Step 3:**
      Project level urls.py using include( ) function

```
from django.contrib import admin
from django.urls import path, include
from App_Two import views as v
urlpatterns = [
    path("admin/", admin.site.urls),
    path("loan/", include('App_Two.urls')),
]
```

- **Templates (Tags)** (need to check with recordings)

  - **Templates** are a key part to understanding how Django really works and interacts with your website.
  - The template will contain the static parts of html page (parts that are always the same)
  - There are **Template tags**, which have their own special syntax.
    - This syntax allows you to inject dynamic content that your Django App's views will produce, effecting the final HTML.

  - **Step 1:**
    It needs to create directory
    >> project_folder / templates / first_app (html)

    - Inside this HTML file we will insert template tags (a.k.a. Django Template Variable.)
    - These template variables will allow us to inject content into the HTML directly from Django.

  - **Step 2:**
    Settings.py >> **TEMPLATES**

    To easily transfer our Django Project one computer to another, the DIR key will require a "hard-coded" path.

```
# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

TEMPLATE_DIR = Path.joinpath(BASE_DIR, 'templates')
```

TEMPLATE_DIR = Path.joinpath(BASE_DIR, 'templates')

- Take that TEMPLATE_DIR and paste in

```
56  TEMPLATES = [
57      {
58          "BACKEND": "django.template.backends.django.DjangoTemplates",
59          "DIRS": [TEMPLATE_DIR],
60          "APP_DIRS": True,
```

- html file inserted in template directory

```
1   <!DOCTYPE html>
2   <html lang="en" dir="ltr">
3     <head>
4       <meta charset="utf-8">
5       <title>Home</title>
6     </head>
7     <body>
8       <h1>Hello this is index.html!</h1>
9       {{insert_me}} {% comment %}Template tag{% endcomment %}
10    </body>
11  </html>
```

Template tag used for injecting data using Django to html which is key of the dictionary.

- BY using render( ) function data injected to html with the help of template tag

```
1   from django.shortcuts import render
2   from django.http import HttpResponse
3   # Create your views here.
4
5   def index(r):
6       my_dict = {'insert_me':'Hello I am from views.py'}
7       return render(r, 'index.html', context=my_dict)
```

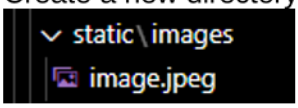
my_dict = { 'insert_me' : 'Hello I am from views.py' } # it's a data in dictionary form
where insert me is a template tag (are same**)

return render(r, 'index.html', context=my_dict)
By using render function data injected in html where context is used for injection.

- After injecting data to html using render function, html result:

- **Static Files** (Media files like CSS, JS, images, etc.)

  - Create a new directory inside of project called static (just like templates)



  - Then add this directory path to the project's settings.py file and STATIC_URL.

```
BASE_DIR = Path(__file__).resolve().parent.parent

TEMPLATE_DIR = Path.joinpath(BASE_DIR, 'templates')
STATIC_DIR = Path.joinpath(BASE_DIR, 'static')
```

STATIC_URL (when we are using URLS for images or CSS or any static file)

For local directory we need insert following list

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.1/howto/static-files/
STATIC_URL = "static/"
STATICFILES_DIRS =[
    STATIC_DIR,
]
```

To check its working or not do following

  - To inject these static files to html page @ we need to add some specific tags, at the top:

{% load static%} # after <! DOCTYPE html>

```
<!DOCTYPE html>
{% load static %}
<html lang="en" dir="ltr">
  <head>
```

  - Then we want to insert the image with an HTML <img src=> using this style tag:

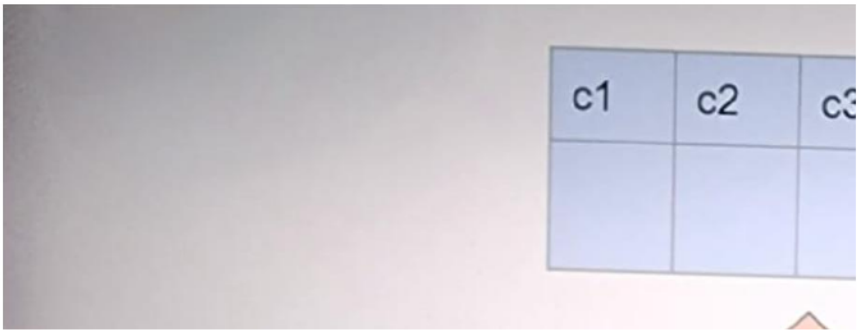
<img src = "{% static "images/image.jpeg" %}" ALT="" >

```
<!DOCTYPE html>
{% load static %}
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Image</title>
  </head>
  <body>
    <h1>This is a static image test</h1>
    <img src="{% static "images/image.jpeg"%}" alt="">

  </body>
</html>

```

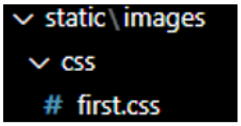
- HTML OUTPUT:

127.0.0.1:8000/help/

This is a static image test



- With CSS




```
h1{
    color: red;
}

```

Link this css to html

```
<!DOCTYPE html>
{% load static %}
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Image</title>
    <link rel="stylesheet" href="{%static "css/first.css"%}"
  </head>
  <body>
    <h1>This is a static image test</h1>
    <img src="{% static "images/image.jpeg"%}" alt="">

  </body>
</html>
```

• Models and Databases

- An essential part of any website is the ability to accept information from a user and input it into a database and retrieve information from a database and use it to generate content for the user.
- We use Models to incorporate a database into a Django Project.
- Django comes equipped with SQLite.
- SQLite will work for our simple projects, but Django can connect to a variety of SQL engine backends
- In the settings.py file you can edit the ENGINE parameter used for DATABASES
- To create an actual model, we use a class structure inside of the relevant applications model.py file
- This class object will be a subclass of Django's built-in class:
    - django.db.models.Model
- Then each attribute of the class represents a field, which is just like a column name with constraints in SQL.

- Inside app directory > **models.py**

```python
from django.db import models

class Topic(models.Model):
    top_name = models.CharField(max_length=264,unique=True)

    def __str__(self): # String representation of the model
        return self.top_name

class WebPage(models.Model) :
    topic = models.ForeignKey(Topic, on_delete=models.CASCADE)
    """categorie = models.ForeignKey(
    'Categorie',
    on_delete=models.CASCADE, #or on_delete=models.DO_NOTHING,
    )"""
    name = models.CharField(max_length=264,unique=True)
    url = models.URLField(unique=True)

    def __str__(self):
        return self.name

class AccessRecord(models.Model):
    name = models.ForeignKey(WebPage, on_delete=models.CASCADE)
    date = models.DateTimeField()

    def __str__(self):
        return str(self.date)

```

Code for creating SQL table(on_delete)

- To execute above code following command is performed: ***
    - **python manage.py migrate**

```
Django version 4.1.5, using settings 'Level_two.settings'
(venv) PS E:\VScode_projects\Django_One_VScode\L_TWO\Level_two> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
```

    - To register migration changes to your application: ***
      python manage.py makemigrations current_app_name

```
Migrations for 'l_two_app':
  l_two_app\migrations\0001_initial.py
    - Create model Topic
    - Create model WebPage
    - Create model AccessRecord
```

- **Repeat after makemigrations to confirm** ***
  python manage.py migrate

- **How to interact with created data base:**
  python manage.py shell (only for testing following its used for this admin interface is used)

```
(venv) PS E:\VScode_projects\Django_One_VScode\L_TWO\Level_two> python manage.py shell
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> print("Hello")
Hello
>>> from l_two_app.models import Topic
>>> print(Topic.objects.all())
<QuerySet []>
>>> x = Topic(top_name="Social Network")
>>> x.save()
>>> print(Topic.objects.all())
<QuerySet [<Topic: Social Network>]>
```

- **For admin interface we have to register model with admin.py**

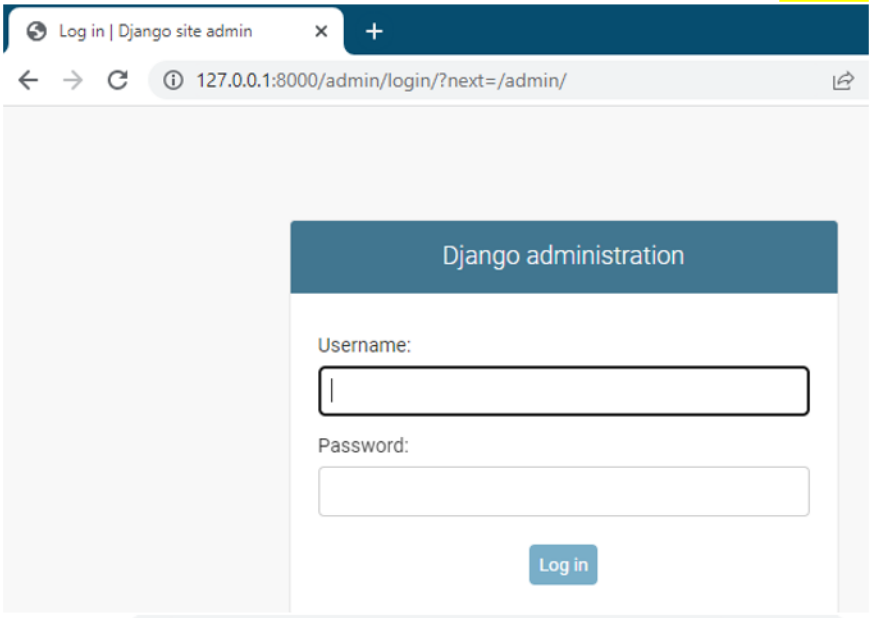
L_TWO > Level_two > l_two_app > admin.py

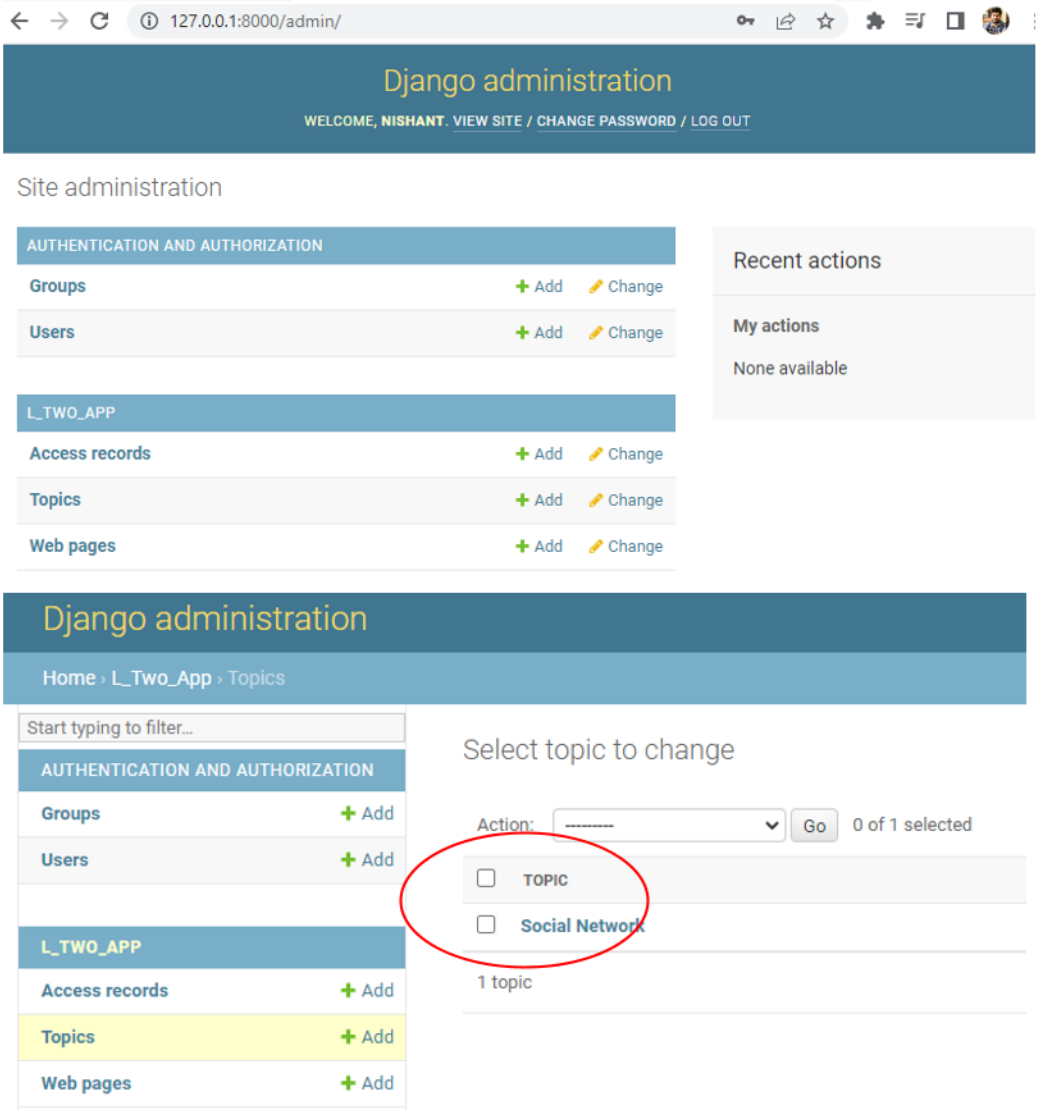
```
1  from django.contrib import admin
2  from l_two_app.models import AccessRecord,Topic,WebPage
3  # Register your models here.
4  admin.site.register(AccessRecord)
5  admin.site.register(Topic)
6  admin.site.register(WebPage)
7
```

- **To create admin super user**
  python manage.py createsuperuser

```
(venv) PS E:\VScode_projects\Django_One_VScode\L_TWO\Level_two> python manage.py createsuperuser

You have 1 unapplied migration(s). Your project may not work properly until you apply the migrati
Run 'python manage.py migrate' to apply them.
Username (leave blank to use 'innk'): nishant
Email address: nishant@gmail.com
Password:
Password (again):
Error: Your passwords didn't match.
Password:
Password (again):
The password is too similar to the username.
This password is too short. It must contain at least 8 characters.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
(venv) PS E:\VScode_projects\Django_One_VScode\L_TWO\Level_two>
```

- **To access admin interface, you need to run server and use URL: admin/**



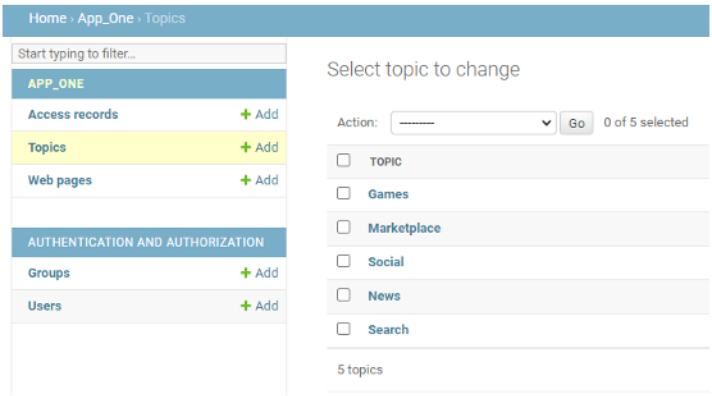


- **Populating Models**

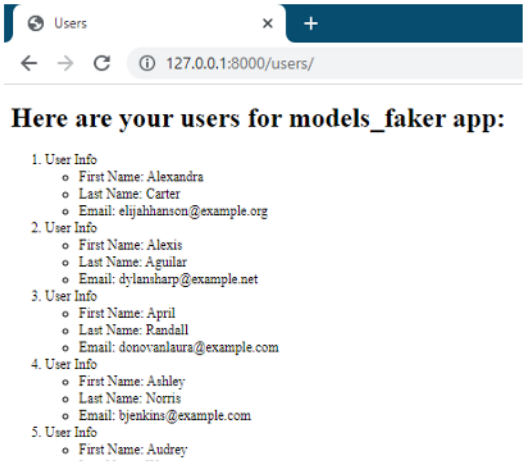
- It's usually a good idea to create a script that will populate your models with some "dummy" data.
- How to use the [Faker library] to create this script.
- **pip install Faker**

```python
import os
# Configure settings for project
# Need to run this before calling models from application!
os.environ.setdefault('DJANGO_SETTINGS_MODULE','Level_two.Level_two.settings')

import django
# Import settings
django.setup()

import random
from .Level_two.l_two_app.models import Topic,WebPage,AccessRecord
from faker import Faker

fakegen = Faker()
topics = ['Search','Social','Marketplace','News','Games']

def add_topic():
    t = Topic.objects.get_or_create(top_name=random.choice(topics))[0]
    t.save()
    return t

def populate(N=5):
    '''
    Create N Entries of Dates Accessed
    '''

    for entry in range(N):

        # Get Topic for Entry
        top = add_topic()

        # Create Fake Data for entry
        fake_url = fakegen.url()
        fake_date = fakegen.date()
        fake_name = fakegen.company()

        # Create new Webpage Entry
        webpg = WebPage.objects.get_or_create(topic=top,url=fake_url,name=fake_name)[0]

        # Create Fake Access Record for that page
        # Could add more of these if you wanted...
        accRec = AccessRecord.objects.get_or_create(name=webpg,date=fake_date)[0]


if __name__ == '__main__':
    print("Populating the databases...Please Wait")
    populate(20)
    print('Populating Complete')

```

Auto generated data



Models-Templates-Views Paradigm (Inserting data for model means from D/B to html)



- **FORMS**

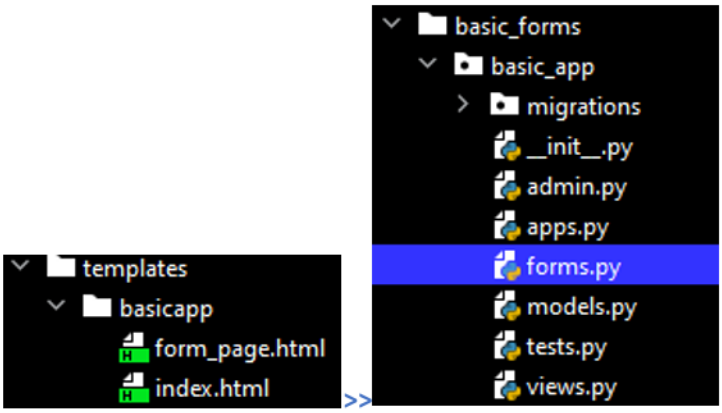
- It's used for accepting user input and connect it to the database and retrieve it letter on.
- Django Forms Advantages:
  - Quickly generate HTML form widgets
  - Validate data and process it into a Python data structure
  - Create form versions of our Models, quickly update models from Forms.
  - We need to create **forms.py** file inside app manually
  - **from django import forms**
  - Inside our views.py file we need to import the forms
  - **from forms import FormName**

- **HTTP | GET | POST | CSRF**
  - **HTTP** stands for Hypertext Transfer Protocol and is designed to enable communication between a client and a server. The client submits a request, the server then responds.

  - **GET** requests data from a resource.

  - **POST** submit data to be process to a resource.

  - **CSRF** is a Cross-Site Request Forgery Token, which is secure the **HTTP POST** action that is initiated on the subsequent submission of form. **{%csrf_token%}**
    - The Django framework requires the CSRF token to be present.
    - If it is not there, your form may not work!
    - It works by using a "hidden input" which is a random code and checking that is matches the user's local site page.
- **Steps**



```python
from django import forms


class FormName(forms.Form):
    name = forms.CharField()
    email = forms.EmailField()
    text = forms.CharField(widget=forms.Textarea)
    # Here widget is passing using Textarea method or attribute
```

forms.py

```python
from django.shortcuts import render
from . import forms


def index(request):
    return render(request, "basicapp/index.html")


def form_name_view(request):
    form = forms.FormName()
    return render(request, "basicapp/form_page.html", {'form': form})
```

views.py

```python
from django.urls import path
from . import views

urlpatterns = [
    path("", views.form_name_view),
]
```

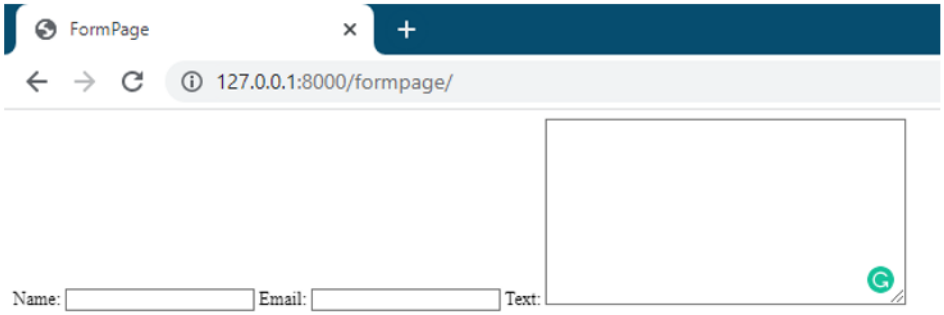
App Level urls.py

```python
from django.contrib import admin
from django.urls import path, include
from basic_app import views

urlpatterns = [
    path("admin/", admin.site.urls),
    path("", views.index),
    path("formpage/", include("basic_app.urls")),
]
```

Project Level urls.py

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>FormPage</title>
</head>
<body>
    <div class = "container">
        {{form}}
    </div>
</body>
</html>
```

form_page.html

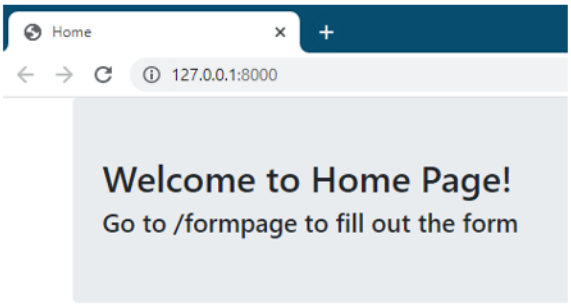


code response

After using bootstrap

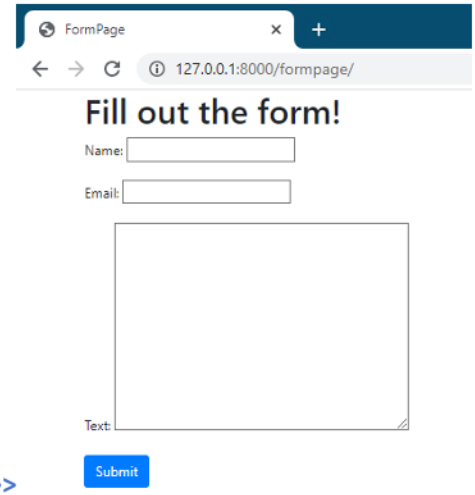
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Home</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapc
    integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApm
</head>
<body>
    <div class="container">
        <div class="jumbotron">
            <h2>Welcome to Home Page!</h2>
            <h3>Go to /formpage to fill out the form</h3>
        </div>
    </div>

</body>
</html>
```

>>

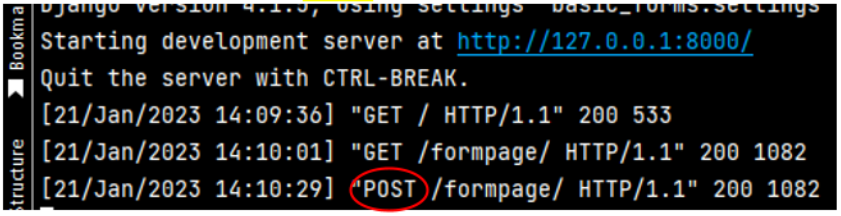



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>FormPage</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bo
    integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoP
</head>
<body>
    <div class="container">
        <h1>Fill out the form!</h1>
      <form method="POST">
          {{ form.as_p }}
          {% csrf_token %}
        <input type="submit" class="btn btn-primary" value="Submit">
      </form>
    </div>
</body>
</html>
```

>>



.as_p (link) also without token form never works with POST or GET



after filling form and submitting

```python
def index(request):
    return render(request, "basicapp/index.html")


def form_name_view(request):
    form = forms.FormName()

    if request.method == 'POST':
        form = forms.FormName(request.POST)

        if form.is_valid():
        # Do somthing Code
        print("Validation Success!")
        print("Name:" +form.cleaned_data['name'])
        print("Email:" +form.cleaned_data['email'])
        print("Text:" +form.cleaned_data['text'])

    return render(request, "basicapp/form_page.html", {'form': form})
```

to check what is submitted in form some modifications

views.py

```
[21/Jan/2023 14:23:21] "GET /formpage/ HTTP/1.1" 200 1082
Validation Success!
Name:nishant
Email:nishant@gmail.com
Text:Hello!!
[21/Jan/2023 14:23:41] "POST /formpage/ HTTP/1.1" 200 1131
```

- **FORM VALIDATION**

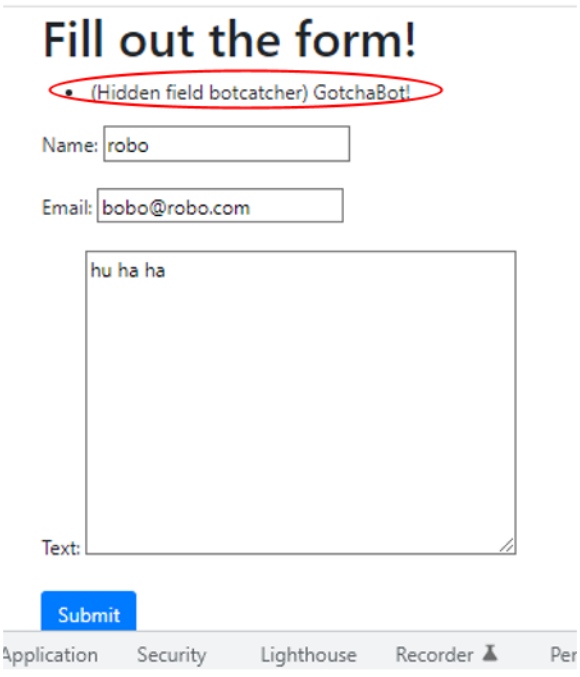
- Above form is accessible for "bots" for misuse of our form.
- Django has built-in validators you can conveniently use to validate your forms (or check for bots!)
    - **Adding a check for empty fields**
    - **Adding a check for a "bot"**
    - **Adding a clean method for the entire form.**

```python
class FormName(forms.Form):
    name = forms.CharField()
    email = forms.EmailField()
    text = forms.CharField(widget=forms.Textarea)
    botcatcher = forms.CharField(required=False,
                                 widget=forms.HiddenInput)
    # required=False is used to hide it from form it works in background

    def clean_botcatcher(self):
        botcatcher = self.cleaned_data['botcatcher']
        if len(botcatcher) > 0:
            raise forms.ValidationError("GotchaBot!")
        return botcatcher
```

manual bot validation

```
" cols="40" rows="10" required id
 name="botcatcher" value="hello"
```

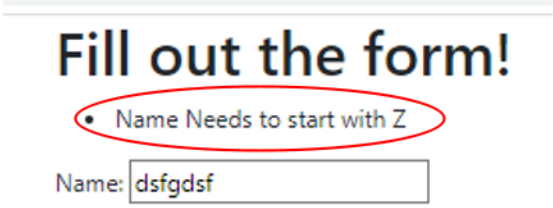
edited inside inspect



```python
from django import forms
from django.core import validators

class FormName(forms.Form):
    name = forms.CharField()
    email = forms.EmailField()
    text = forms.CharField(widget=forms.Textarea)
    botcatcher = forms.CharField(required=False,
                                 widget=forms.HiddenInput,
                                 validators=[validators.MaxLengthValidator(0)])
```

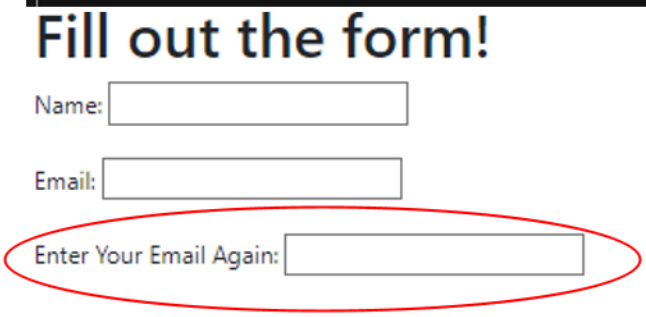
Built-in validator

```python
from django import forms
from django.core import validators

def check_for_z(value):
    if value[0].lower() != 'z':
        raise forms.ValidationError("Name Needs to start with Z")

class FormName(forms.Form):
    name = forms.CharField(validators=[check_for_z])
    email = forms.EmailField()
    text = forms.CharField(widget=forms.Textarea)
    botcatcher = forms.CharField(required=False,
                                 widget=forms.HiddenInput,
                                 validators=[validators.MaxLengthValidator(0)])
```



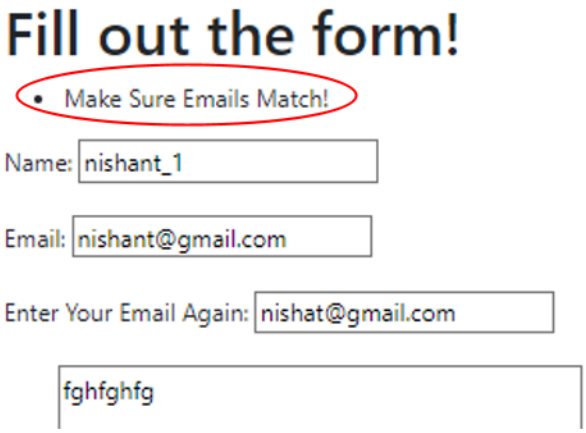
- You can make your own validator as per the requirement using documentations.

```python
from django import forms


class FormName(forms.Form):
    name = forms.CharField()
    email = forms.EmailField()
    verify_email = forms.EmailField(label="Enter Your Email Again")
    text = forms.CharField(widget=forms.Textarea)
```



- **Clean Method**

```python
from django import forms


class FormName(forms.Form):
    name = forms.CharField()
    email = forms.EmailField()
    verify_email = forms.EmailField(label="Enter Your Email Again")
    text = forms.CharField(widget=forms.Textarea)

    def clean(self):
        all_clean_data = super().clean()
        email = all_clean_data['email']
        verify_email = all_clean_data['verify_email']

        if email != verify_email:
            raise forms.ValidationError("Make Sure Emails Match!")
```



- **Models Forms**

- Saving data from Form to models (accepting Form input and passing it to a model)
- Instead of inheriting from the **forms.Forms** class, we will use **forms.ModelForm** in forms.py file.
- This helper class allows us to create a form a pre-existing **model**. We need to add an inline class called **Meta.** This **Meta** class provided information connecting the **model** to the **Form**.

- Option #1: Set it to " _ _all_ _"

```
from django import forms
from myapp.models import MyModel
class MyNewForm(forms.ModelForm):
        # Form Fields go here
        class Meta:
                model = MyModel
                fields = "_ _all_ _"  # Its grabbing all the fields form the model place that into the Form
```

- Option #2: Exclude certain fields

```
from django import forms
from myapp.models import MyModel
class MyNewForm(forms.ModelForm):
        # Form Fields go here
        class Meta:
                model = MyModel
                fields = ["field_1, field_2"] # Given field excluded
```

- Option #3: List included fields

```
from django import forms
from myapp.models import MyModel
class MyNewForm(forms.ModelForm):
        # Form Fields go here
        class Meta:
                model = MyModel
                fields = (“field_1, field_2”) # In some cases huge amount of fields are present to select as per requirement (“field_1, field_2”) is used.
```

```python
from django import forms
from .models import User

class NewUserForm(forms.ModelForm):
    class Meta:
        model = User
        fields = '__all__'
```

forms.py

```python
from django.db import models


class User(models.Model):
    first_name = models.CharField(max_length=128)
    last_name = models.CharField(max_length=128)
    email = models.EmailField(max_length=256, unique=True)
```

modesl.py(migration performed)

```python
from django.shortcuts import render
from .forms import NewUserForm


def index(request):
    return render(request, 'home.html')


def users(request):
    form = NewUserForm()

    if request.method == "POST": # submit button executed
        form = NewUserForm(request.POST) # We sending request.POST

        if form.is_valid(): # If the given info is valid
            form.save(commit=True) # save that
            return index(request) # return to index(Home Page)

        else:
            print('Error Form Invalid')

    return render(request, 'users.html', {'form': form})
```

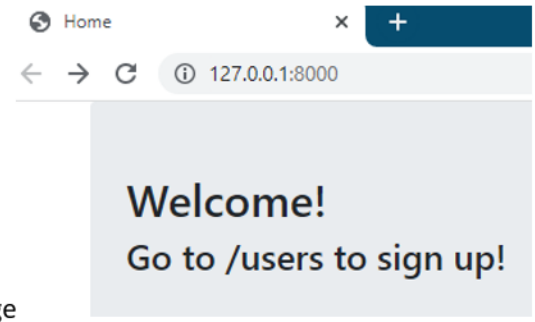
views.py (with form validation)

```python
from django.urls import path
from . import views

urlpatterns = [
    path("", views.users),
]
```

app level urls.py

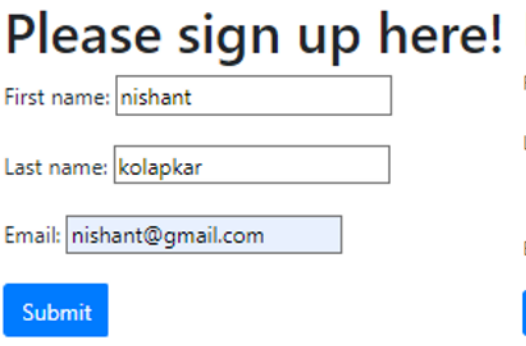
```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Home</title>
    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet" href="https://stac
    integrity="sha384-MCw98/SFnGE8fJT3GXwEOng

  </head>
  <body>
    <div class="container">
      <div class="jumbotron">
        <h1>Welcome!</h1>
        <h2>Go to /users to sign up!</h2>
      </div>
    </div>

  </body>
</html>
```

Home page



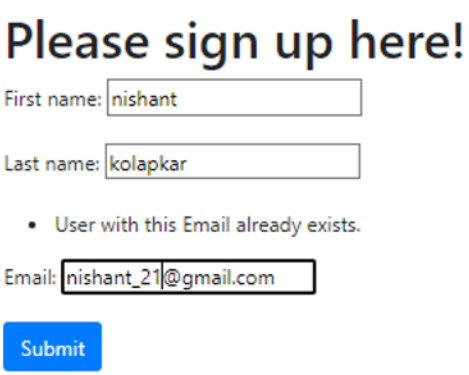

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Users</title>
    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/b
    integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXo

  </head>
  <body>
    <div class="container">
      <h1>Please sign up here!</h1>
      <form method="POST">
        {{ form.as_p }} # converts each field of Form as paragraph
        {% csrf_token %}
        <input type="submit" class='btn btn-primary' value="Submit">
      </form>
    </div>

  </body>
</html>
```

Sign Up page

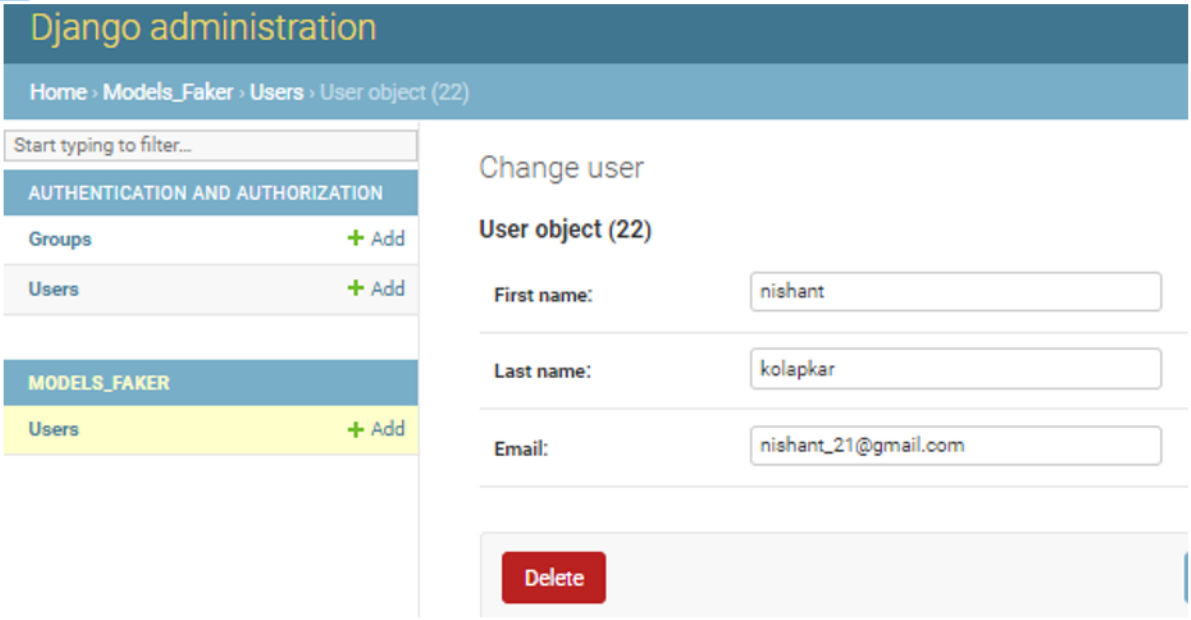


Validation working

```
[22/Jan/2023 16:03:17] "GET /users/ HTTP/1.1" 200 1235
Error Form Invalid
```



Successfully submitted and return to home page.



- **"Base" Template and Inheriting that template into the .html file**
- **Relative URLs with Templates**

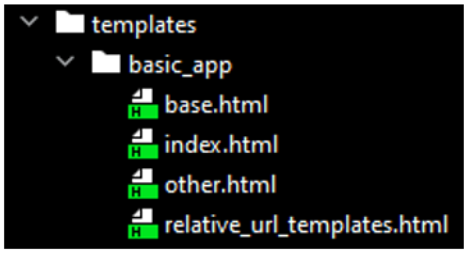
- Before Template tagging, we used **anchor** tag with an **href** we've passed in a hardcoded path to the file.
- This is poor practice if we want our Django project to work on any system.

- We can easily fix this with the use of URLs in our templates. For example:
  Thanks
  Can be changed to:
  Thanks
  name='thanku' is in the **urls.py** file.

- Direct referencing views is also works
  Thanks
  Can be changed to:
  Thanks

- This is the best way to use URL templates:
  Thanks
  Can be changed to:
  Thanks
  This method requires that app_name variable to be created inside the urls.py file.

- **STEPS**




```python
from django.shortcuts import render

def index(r):
    return render(r, "basic_app/index.html")

def other(r):
    return render(r, "basic_app/other.html")

def relative(r):
    return render(r, "basic_app/relative_url_templates.html")
```

basic_app/views.py

```python
from django.urls import path
from . import views

# Template Tagging
app_name = 'basic_app'

urlpatterns = [
    path("other/", views.other, name="other"),
    path("relative/", views.relative, name="relative"),
]
```

app level urls.py

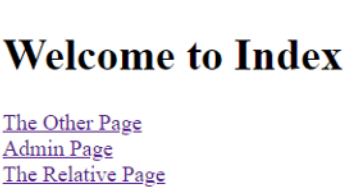
```python
from django.contrib import admin
from django.urls import path, include
from basic_app import views
urlpatterns = [
    path("admin/", admin.site.urls),
    path("", views.index, name="index"),
    path("", include("basic_app.urls")),
```

project level urls.py

Note: To access admin page from URLs, you have to run python manage.py migrate then it works.

```html
    <title>Index</title>
</head>
<body>
    <h1>Welcome to Index</h1>
    <a href="{% url 'basic_app:other'%}">The Other Page</a>
    <br>
    <a href="{%url 'admin:index' %}">Admin Page</a>
    <br>
    <a href="{% url 'basic_app:relative' %}">The Relative Page</a>
```

>>



```
    <title>Other</title>
</head>
<body>
    <h1>Welcome to Other</h1>
     <a href="{% url 'basic_app:relative' %}">The Relative Page</a>
    <br>
    <a href="{%url 'index' %}">Index Page</a>
```

>>

Welcome to Other

The Relative Page
Index Page

```
    <title>Relative</title>
</head>
<body>
    <h1>Welcome to relative urls template</h1>
    <a href="{% url 'basic_app:other'%}">The Other Page</a>
    <br>
    <a href="{%url 'admin:index' %}">Admin Page</a>
    <br>
    <a href="{%url 'index' %}">Index Page</a>
</body>
</html>
```

>>

Welcome to relative

The Other Page
Admin Page
Index Page

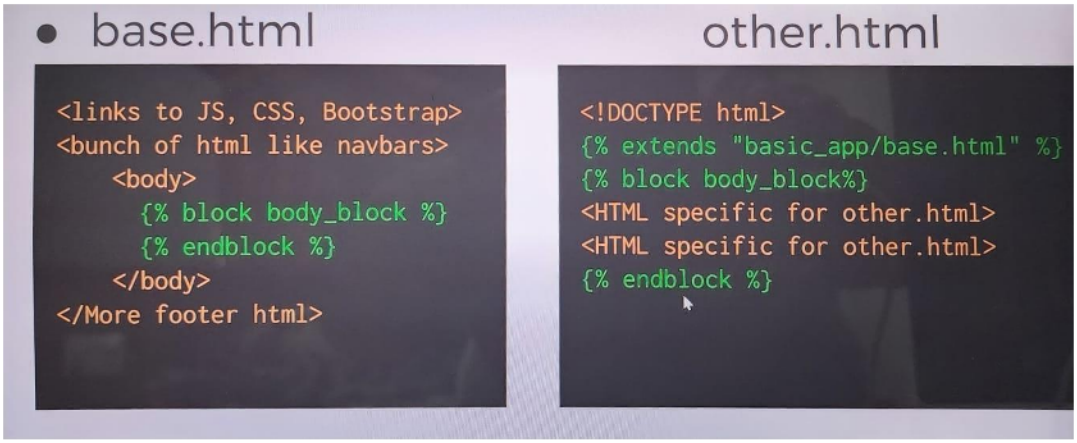
- **Template Inheritance**

- How we can use Django Template Inheritance to practice DRY coding principle

Don't repeat yourself

"Don't repeat yourself" is a principle of software development aimed at reducing repetition of software patterns, replacing it with abstractions or using data normalization to avoid redundancy. Wikipedia

- Template inheritance allows us to create a base template we can inherit from.
- This saves us a lot of repetitive work and makes it much easier to maintain the same base look and feel across our entire website.
- Before you being any Django Project, it is always a good idea to sketch out the main idea and organization by hand. This will help you realize what can be used for template inheritance and what application you should create.
- Main Steps for Inheritance:
    - Find the repetitive parts of your project
    - Create a base template of them
    - Set the tags in the base template
    - Extend and call those tags anywhere

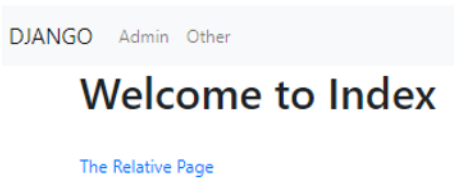


```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Base</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1
    integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdkr
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
      <div class="navbar-nav">
        <a class="navbar-brand" href="{% url 'index' %}">DJANGO</a>
        <a class="nav-item nav-link" href="{% url 'admin:index' %}">Admin</a>
        <a class="nav-item nav-link" href="{% url 'basic_app:other' %}">Other</a>
      </div>
    </nav>
    <div class="container">
        {% block body_block %}
        <!--Anything outside of this will be inherited if you use extend.-->
        {% endblock %}
    </div>
</body>
</html>
```

base.html

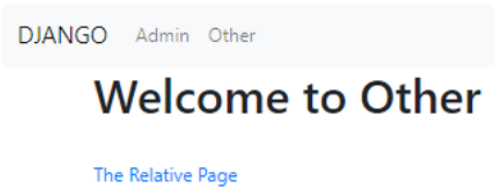
```html
<!DOCTYPE html>
<head>
    <meta charset="UTF-8">
    <title>Home</title>
</head>

{% extends "basic_app/base.html" %}

    {% block body_block %}
        <h1>Welcome to Index</h1>
        <br>
        <a href="{% url 'basic_app:relative' %}">The Relative Page</a>
    {% endblock %}
```

index.html(inherited with base.html)




```html
<!DOCTYPE html>
<head>
    <meta charset="UTF-8">
    <title>Other</title>
</head>

{% extends "basic_app/base.html" %}

    {% block body_block %}
        <h1>Welcome to Other</h1>
        <a href="{% url 'basic_app:relative' %}">The Relative Page</a>
        <br>
        <a href="{%url 'index' %}">Index Page</a>
    {% endblock %}
```

other.html(inherited with base.html)



- **Template Filters and Custom Filters (Link)**

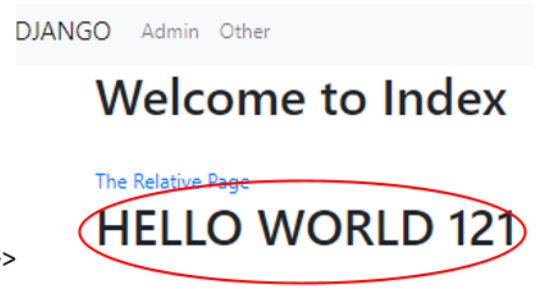
- The general form for a template filter is:
  - {{ value|filter: “parameter” }} (| pipe operator with no space)
  - Not all filters take in parameters.

```python
def index(r):
    context_dict = {'text': 'Hello world', 'number': 100}
    return render(r, "basic_app/index.html", context_dict)
```

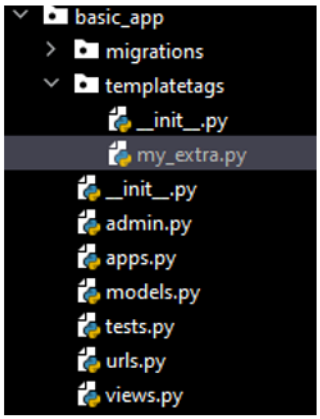
views.py

```
<h1>
    {{text|upper}}
    {{number|add:"21"}}</h1>
```

>>



- **Custom template filter**



for Custom Template Filter we have to make this dir inside app with __init__.py and custom_filter.py

```python
from django import template

register = template.Library()


def cut(value, arg):
    """This cuts out all values of 'arg' from the string"""
    return value.replace(arg, '')  # its common python string operation


register.filter('cut', cut)
```

Register custom tag my_extra.py

```python
from django import template

register = template.Library()

@register.filter(name='cut')
def cut(value, arg):
    """This cuts out all values of 'arg' from the string"""
    return value.replace(arg, '')  # its common python string operation
```

registering by using Decorators | same as above

```
<!DOCTYPE html>
<head>
    <meta charset="UTF-8">
    <title>Home</title>
</head>

{% extends "basic_app/base.html" %}

    {% block body_block %}
        {% load my_extra %}
        <h1>Welcome to Index</h1>
        <br>
        <a href="{% url 'basic_app:relative' %}">The Relative Page</a>
        <br>
        <h1>
            {{text|cut:'Hell'}}
            {{number|add:"21"}}</h1>
    {% endblock %}
```

loading custom tag

- **Django Passwords (Documentation)**

```
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
```

- **Never store passwords as plain text!**
- We will begin by using the default **PBKDF2** algorithm with an **SHA256** hash that is built-in to Django.
- **pip install bcrypt**
- **pip install django[argon2]**
- **SHA-256 HAS calculator (link)**
- After installing above two algo in venv
- Need to pass **password hashers** list into settings.py

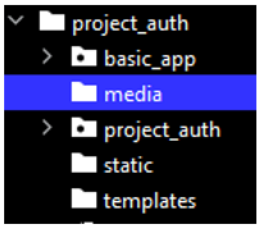
```
# Password validation
# https://docs.djangoproject.com/en/4.1/ref/settings/#auth-pass

PASSWORD_HASHERS = [
    'django.contrib.auth.hashers.PBKDF2PasswordHasher',
    'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',
    'django.contrib.auth.hashers.BCryptPasswordHasher',
    'django.contrib.auth.hashers.PBKDF2PasswordHasher',
    'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',
]
```

```
AUTH_PASSWORD_VALIDATORS = [
    {
        "NAME": "django.contrib.auth.password_validation.UserAttributeSimilarityValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.MinimumLengthValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.CommonPasswordValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.NumericPasswordValidator",
    },
]
```

password validation dictionary

```
AUTH_PASSWORD_VALIDATORS = [
    {
        "NAME": "django.contrib.auth.password_validation.UserAttributeSimilarityValidator",
    },
    {
        "NAME": "django.contrib.auth.password_validation.MinimumLengthValidator",
        'OPTIONS': {'min_length': 9}
```

Lots of options are available in documentation




```
STATIC_DIR = Path.joinpath(BASE_DIR, "static")
MEDIA_DIR = Path.joinpath(BASE_DIR, "media")
```

```
STATIC_URL = "static/"
STATICFILES_DIRS = [
    STATIC_DIR,
]
MEDIA_ROOT = MEDIA_DIR
MEDIA_URL = "media/"
```

media directory is used for user data.

- **User Models** (How to use Django's built in tools to create User Authrization Models)

- As Admin we seen Admin page which is already built-in Authentication and Authorization model set in place.

- Now see as **"Users"**
- The **User** object has a few key features:
    - Username
    - Email
    - Password
    - First Name
    - Surname
- There are also some other attributes for the User object, such as **is_active, is_staff, is_superuser**. Sometimes you will also want to add more attribute to a user, such as their own links or a profile image.
- We can do above all in applications **models.py** file by creating another class that has a relationship to the **User class**.
- In order to work with images with Python we will need to install the Python Imaging Library with:
  pip install pillow

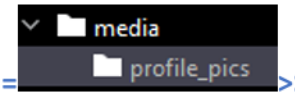
- **Forms** are used for collecting data from **Users**.

- **Coding User Models and Forms**

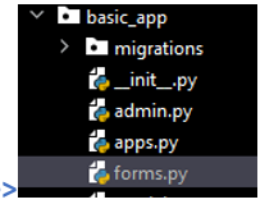
```python
from django.db import models
from django.contrib.auth.models import User
# Create your models here.

class UserProfileInfo(models.Model):

    user = models.OneToOneField(User, on_delete=models.CASCADE)
    # User class have all basic fields to modify it OneToOne is used.

    # additional
    portfolio_site = models.URLField(blank=True)

    profile_pic = models.ImageField(upload_to='profile_pics', blank=True)

    def __str__(self):
        return self.user.username
```

models.py

upload_to =



>>>>




```python
from django import forms
from django.contrib.auth.models import User
from .models import UserProfileInfo


class UserForm(forms.ModelForm):
    password = forms.CharField(widget=forms.PasswordInput())

    class Meta():
        model = User
        fields =('username', 'email', 'password')


class UserProfileInfoForm(forms.ModelForm):
    class Meta():
        model = UserProfileInfo
        fields = ("portfolio_site", "profile_pic")
```

forms.py

```python
from django.contrib import admin
from .models import UserProfileInfo
# Register your models here.
admin.site.register(UserProfileInfo)
```

admin.py **models** are registered here

- **Connecting to HTML file**

- A lot of coding for working with Users and Authorization happens in the views.py
- The basic idea is that we check if there is a POST request and then perform some sort of action based off that information.
- Sometimes we will want to save that information directly to the database.

- Other times, we will set commit = False so we can manipulate the data before saving(save()) it to the database.
- This helps prevent collision errors.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title></title>
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.
    integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERd
</head>
<body>
  <nav class="navbar navbar-expand-lg navbar-light bg-light">
     <div class="navbar-nav">
<!--        Django Home Link / Admin Link / Register Link-->
        <a class="navbar-brand" href="{% url 'index' %}">DJANGO</a>
        <a class="nav-item nav-link" href="{% url 'admin:index' %}">Admin</a>
        <a class="nav-item nav-link" href="{% url 'basic_app:register' %}">Regis

<!--        Some logic on what to display for last item-->
        {% if user.is_authenticated %} <!--from views.py-->
          <a class="nav-link" href="{% url 'logout' %}">Logout</a>
        {% else %}
          <a class="nav-link" href="{% url 'basic_app:user_login' %}">Login</a>
        {% endif %}
      </div>
    </nav>

    <div class="container">
    {% block body_block %}
    {% endblock %}
    </div>
</body>
</html>
```

base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Home</title>
</head>
{% extends "basic_app/base.html" %}
    {% block body_block %}

        <div class="container">
          <div class="jumbotron">
            <h1>Welcome to Home page</h1>
            {% if user.is_authenticated %}
                <h2>Welcome {{ user.username }}!</h2>
            {% else %}
                <h2>Let's get started</h2>
            {% endif %}
          </div>
        </div>

    {% endblock %}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Registration</title>
</head>
{% extends "basic_app/base.html" %}


{% block body_block %}
  <div class="container">
    <div class="jumbotron">
      {% if registered %}
        <h1>Thank you for registering!</h1>
      {% else %}
        <h1>Register Here</h1>
        <h3>Just fill out the form.</h3>

        <form enctype="multipart/form-data" method="POST">
          {% csrf_token %}
          {{ user_form.as_p }}
          {{ profile_form.as_p }}
          <input type="submit" name="" value="Register">
        </form>

      {% endif %}

    </div>

  </div>

{% endblock %}
```

register.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Login</title>
</head>
{% extends 'basic_app/base.html' %}
{% block body_block %}
  <div class="container">
    <div class="jumbotron">
      <h1>Please Login</h1>

      <form method="post" action="{% url 'basic_app:user_login' %}">
        {% csrf_token %}
<!--        A more "HTML" way of creating the login form-->
        <label for="username">Username:</label>
        <input type="text" name="username" placeholder="Username">

        <label for="password"></label>
        <input type="password" name="password">

        <input type="submit" name="" value="Login">

      </form>

    </div>
  </div>
{% endblock %}
```

login.html

views.py

```python
from django.shortcuts import render
from .forms import UserForm, UserProfileInfoForm

from django.urls import reverse
from django.contrib.auth.decorators import login_required
from django.contrib.auth import authenticate, login, logout
from django.http import HttpResponseRedirect, HttpResponse

def index(r):
    return render(r, 'basic_app/index.html')


@login_required
def user_logout(r):
    logout(r)
    return HttpResponseRedirect(reverse('index'))

@login_required
def special(r):
    return HttpResponse("You are logged in, Nice!")

def register(r):

    registered = False

    if r.method == "POST":
        user_form = UserForm(data=r.POST)# username,password
        profile_form = UserProfileInfoForm(data=r.POST)#profile_pic

        if user_form.is_valid() and profile_form.is_valid():

            user = user_form.save() # saving username to DB
            user.set_password(user.password)
            user.save() # saving user with HASH password

            profile = profile_form.save(commit=False)
            profile.user = user # user from model.py, profile_pic save with user.

            if 'profile_pic' in r.FILES: # # profile_pic from forms.py
                profile.profile_pic = r.FILES['profile_pic']
                # profile_pic from model. saving to DB

            profile.save()

            registered = True

        else:
            print(user_form.errors,profile_form.errors)
    else:
        user_form = UserForm()
        profile_form = UserProfileInfoForm()

    return render(r, "basic_app/register.html",
                  {'user_form': user_form,
                   'registered': registered,
                   'profile_form': profile_form})

def user_login(r):

    if r.method == 'POST':
        username = r.POST.get('username') # name = username from login.html
        password = r.POST.get('password')

        user = authenticate(username=username, password=password)

        if user:
            if user.is_active:
                login(r, user)
                return HttpResponseRedirect(reverse('index'))

            else:
                return HttpResponse("Account not Active")
        else:
            print("Someone tried tp login and faild!")
            print(f"Username: {username} and password: {password}")
            return HttpResponse("Invalid login details supplied!")
    else:
        return render(r, "basic_app/login.html", {})
```

app level **urls.py**

```python
from django.urls import path
from . import views as v

app_name = 'basic_app' # template tag

urlpatterns = [
    path("register/", v.register, name="register"),
    path("user_login/", v.user_login, name="user_login"),
]
```

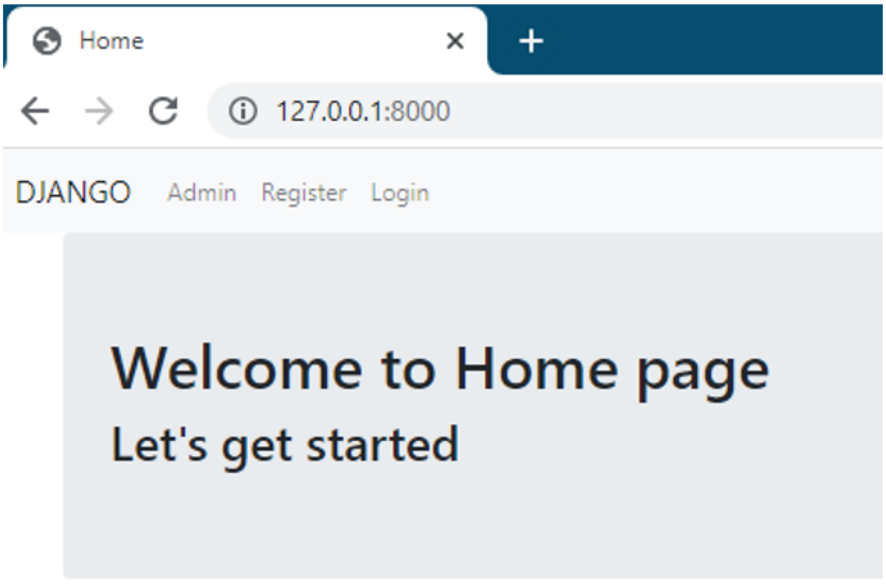
project level **urls.py**

```python
from django.contrib import admin
from django.urls import path, include
from basic_app import views as v

urlpatterns = [
    path("admin/", admin.site.urls),
    path("", v.index, name='index'),
    path("basic_app/", include("basic_app.urls")),
    path("logout/", v.user_logout, name='logout'),
    path("special/", v.special, name='special'),
]
```

for Django's built in login log out functionality we have to set following in **settings.py**

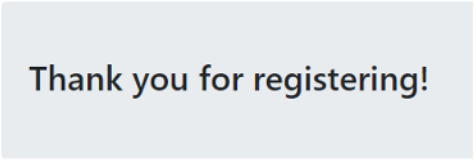
```python
STATIC_URL = "static/"
STATICFILES_DIRS = [
    STATIC_DIR,
]
MEDIA_ROOT = MEDIA_DIR
MEDIA_URL = "media/"

LOGIN_URL = 'basic_app/user_login'
```

Complete OUT PUT:

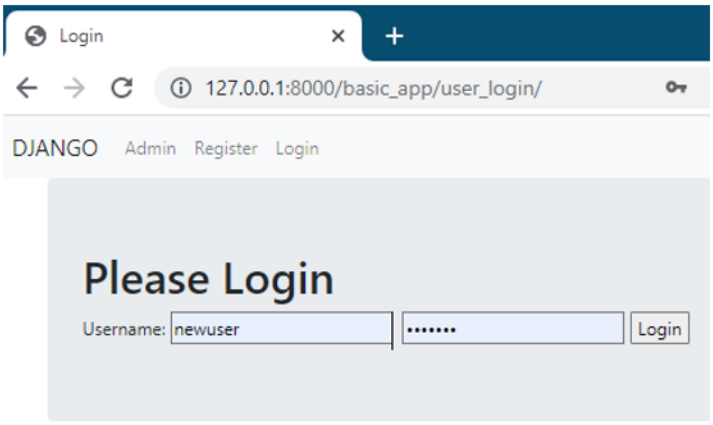


Home page

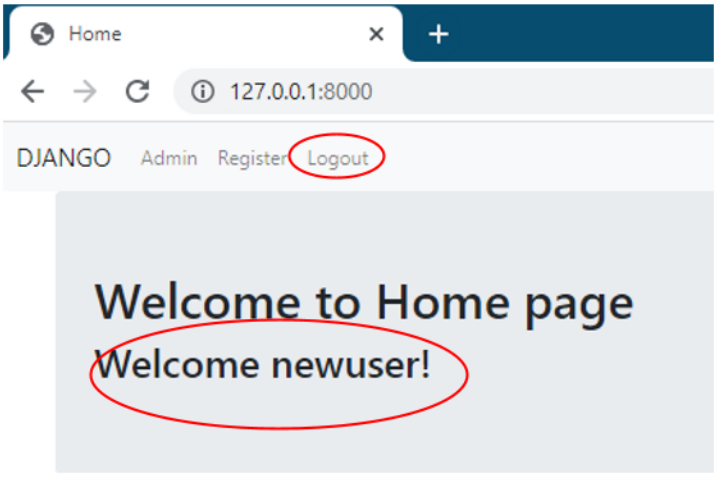
Register page



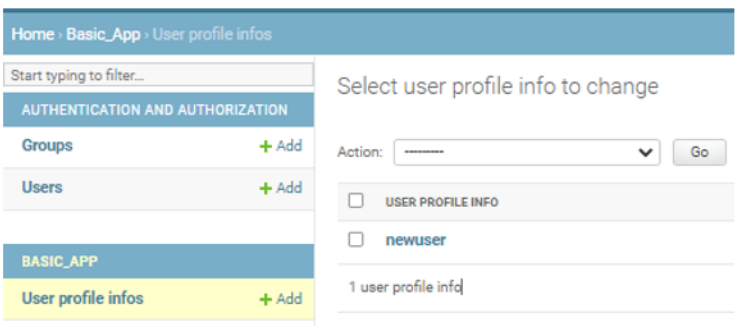
After Registering



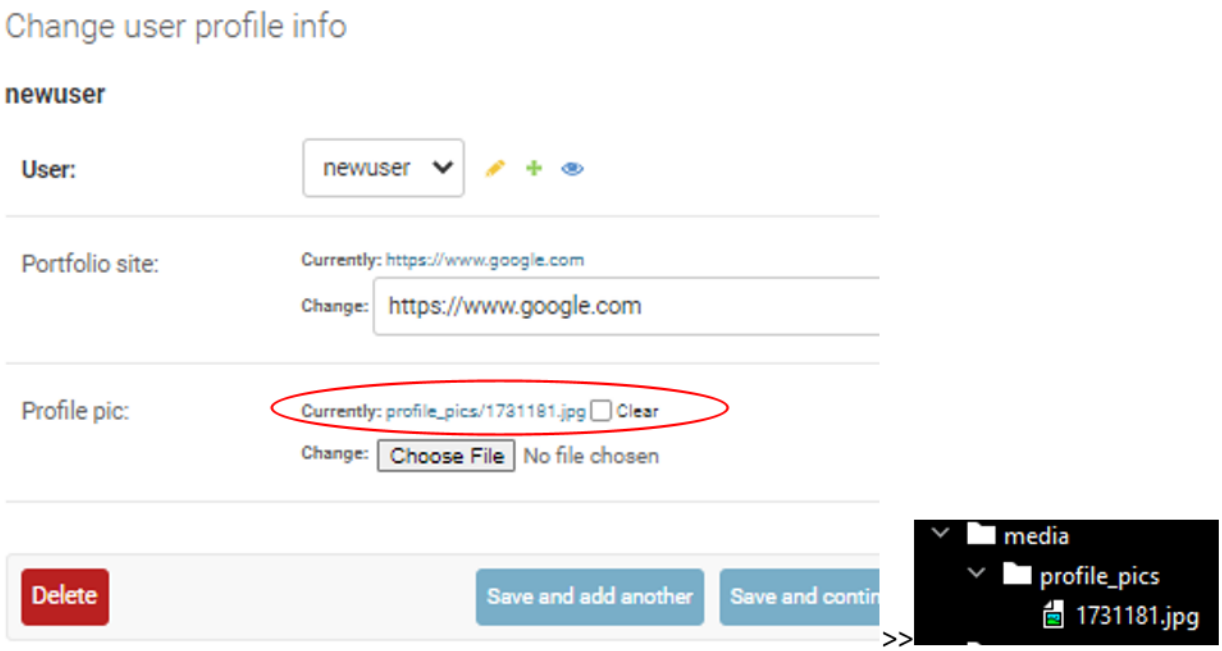
Login page



After login



Inside admin panel



registered user with profile_pic which is stored inside DB.

- Deployment
  - We need to apply some changes in settings.py

```
DEBUG = True

ALLOWED_HOSTS = []
```

DEBUG = False
ALLOWD_HOSTS = [‘username.pythonanywhere.com’]

- **CBV** (Class Based Views) advance Django

```
from django.shortcuts import render

# Create your views here.

def index(r):
    return render(r, 'basic_app/index.html')
```

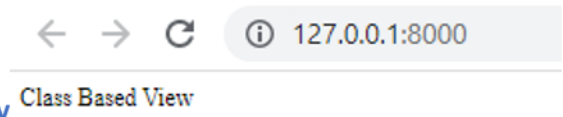
function based **views.py**

```
from django.views.generic import View
from django.http import HttpResponse


class CBView(View):
    def get(self,r):
        return HttpResponse("Class Based View")
```

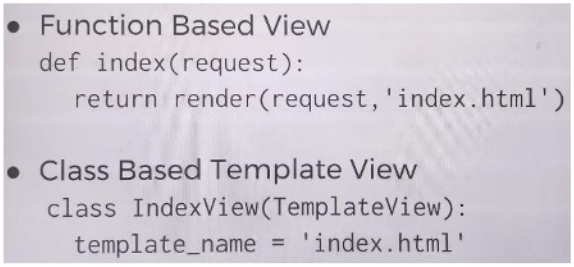
By using **class View inside views.py**

```
from django.contrib import admin
from django.urls import path
from basic_app import views as v
urlpatterns = [
    path("admin/", admin.site.urls),
    path("", v.CBView.as_view()),
]
```

urls.py



- **TemplateView**



function based view vs class based TemplateView

```python
from django.shortcuts import render
from django.views.generic import TemplateView


class IndexView(TemplateView):
    template_name = 'basic_app/index.html'
```

views.py

```python
from django.contrib import admin
from django.urls import path
from basic_app import views as v
urlpatterns = [
    path("admin/", admin.site.urls),
    path("", v.IndexView.as_view()),
```

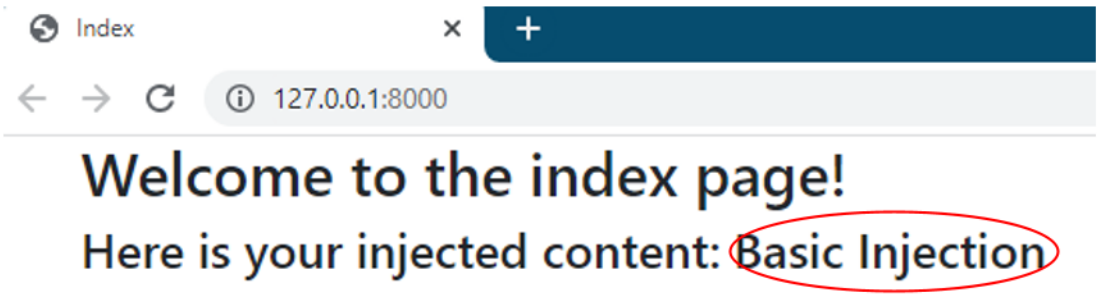
urls.py

- **Injecting context to html (*args **kwargs)**

```python
from django.views.generic import TemplateView


class IndexView(TemplateView):
    template_name = 'basic_app/index.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['injectme'] = 'Basic Injection'
        return context
```

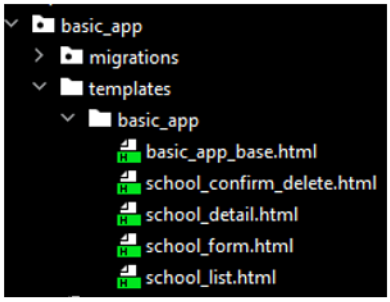
>>

```
    <title>Index</title>
{% extends "basic_app/base.html" %}

  {% block body_block %}
    <h1>Welcome to the index page!</h1>
    <h2>Here is your injected content: {{ injectme }}</h2>
  {% endblock %}
```



- **List Views and Detail View**

- When we have models, we want to either list the records from the model, or show details of single record.
- Previously we did this with calls using the Object Relation Mapper directly.
- This included things like:
  MyModel.object.all( )
- So common that Django has some generic view classes you can inherit to very quickly display information from your model.
- This is where the power of CBV comes to help us out!
  - **CRUD Views** (Create Retrieve Update Delete)



app level templates

- VIEWS, TEMPLATE, LIST, DETAIL, CREATE, DELETE, UPDATE View views.py

```
from django.urls import reverse_lazy
from django.http import HttpResponse
from django.views.generic import (View,TemplateView,
                                  ListView,DetailView,
                                  CreateView,DeleteView,
                                  UpdateView)
from . import models
# Create your views here.
# Original Function View:
# def index(request):
#     return render(request,'index.html')
# Pretty simple right?
class IndexView(TemplateView):
    # Just set this Class Object Attribute to the template page.
    # template_name = 'app_name/site.html'
    template_name = 'index.html'

    def get_context_data(self,**kwargs):
        context  = super().get_context_data(**kwargs)
        context['injectme'] = "Basic Injection!"
        return context

class SchoolListView(ListView):
    # If you don't pass in this attribute,
    # Django will auto create a context name
    # for you with object_list!
    # Default would be 'school_list'
    # Example of making your own:
    # context_object_name = 'schools'
    model = models.School


class SchoolDetailView(DetailView):
    context_object_name = 'school_details'
    model = models.School
    template_name = 'basic_app/school_detail.html'


class SchoolCreateView(CreateView):
    fields = ("name","principal","location")
    model = models.School


class SchoolUpdateView(UpdateView):
    fields = ("name","principal")
    model = models.School

class SchoolDeleteView(DeleteView):
    model = models.School
    success_url = reverse_lazy("basic_app:list")


class CBView(View):
    def get(self,request):
        return HttpResponse('Class Based Views are Cool!')
```

models.py

```
from django.db import models
from django.urls import reverse

# Create your models here.
class School(models.Model):
    name = models.CharField(max_length=256)
    principal = models.CharField(max_length=256)
    location = models.CharField(max_length=256)

    def __str__(self):
        return self.name

    def get_absolute_url(self):
        return reverse("basic_app:detail",kwargs={'pk':self.pk})

class Student(models.Model):
    name = models.CharField(max_length=256)
    age = models.PositiveIntegerField()
    school = models.ForeignKey(School,related_name='students',on_delete=models.CASCADE)

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

admin.py

```
from django.contrib import admin
from .models import School,Student
# Register your models here.
admin.site.register(School)
admin.site.register(Student)
```

app_level urls.py

```
from django.urls import path
from . import views

app_name = 'basic_app'

urlpatterns = [
    path('',views.SchoolListView.as_view(),name='list'),
    path('<int:pk>/',views.SchoolDetailView.as_view(),name='detail'),
    path('create/',views.SchoolCreateView.as_view(),name='create'),
    path('update/<int:pk>/',views.SchoolUpdateView.as_view(),name='update'),
    path('delete/<int:pk>/',views.SchoolDeleteView.as_view(),name='delete')
]
```

project_level urls.py

```
from django.contrib import admin
from django.urls import path, include
from basic_app import views

urlpatterns = [
    path('',views.IndexView.as_view()),
    path('admin/',admin.site.urls,name='admin'),
    path('basic_app/',include('basic_app.urls',namespace='basic_app')),
    # path('',views.CBView.as_view()),
    # path('',views.index)
```

Project_level base.html

```
<!DOCTYPE html>
{% load static %}
<html>
  <head>
    <meta charset="utf-8">
    <title>
      CBVs
      {# Title Extensions go inside the block#}
      {% block title_block %}

      {% endblock %}
    </title>

    {# Bootstrap and CSS (Probably would want downloaded files in your real projects)#}
    {# https://bootswatch.com/#}
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
    integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous">

</head>
  <body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
      <div class="navbar-nav">
        <a class="navbar-brand" href="{% url 'basic_app:list' %}">Schools</a>
        <a class="nav-item nav-link" href="{% url 'admin:index' %}">Admin</a>
        <a class="nav-item nav-link" href="#"></a>
      </div>
    </nav>

    <div class="container">
      {% block body_block %}

      {% endblock %}
    </div>

  </body>

  {# Plugins#}


  <!-- Latest compiled and minified jQuery -->
  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-
q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>
  <!-- Latest compiled and minified JavaScript -->
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"
integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy"
crossorigin="anonymous"></script>

</html>
```

Project_level index.html

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

  {% block body_block %}
    <h1>Welcome to the index page!</h1>
    <h2>Here is your injected content: {{ injectme }}</h2>
  {% endblock %}
```

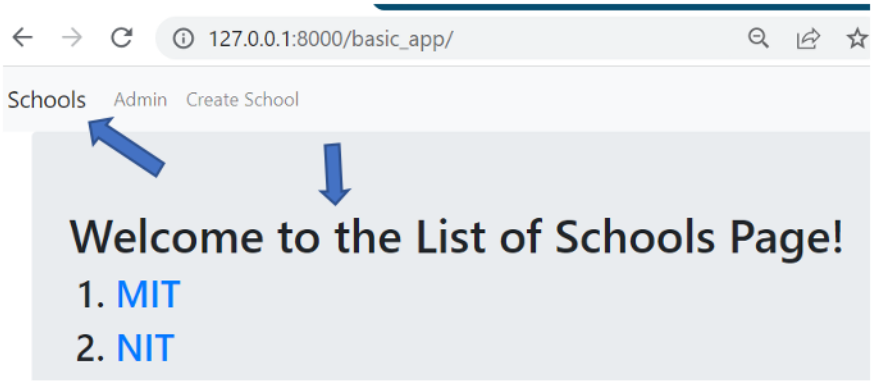
App_level templates

basic_app_base.html

```
<!DOCTYPE html>
{% load static %}
<html>
  <head>
    <meta charset="utf-8">
    <title>
      CBVs
      {# Title Extensions go inside the block#}
      {% block title_block %}

      {% endblock %}
    </title>

    {# Bootstrap and CSS (Probably would want downloaded files in your real projects)#}
    {# https://bootswatch.com/#}
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
    integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
crossorigin="anonymous">

</head>
  <body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
      <div class="navbar-nav">
        <a class="navbar-brand" href="{% url 'basic_app:list' %}">Schools</a>
        <a class="nav-item nav-link" href="{% url 'admin:index' %}">Admin</a>
        <a class="nav-item nav-link" href="{% url 'basic_app:create' %}">Create School</a>
      </div>
    </nav>

    <div class="container">
      {% block body_block %}

      {% endblock %}
    </div>

  </body>

  {# Plugins#}

  <!-- Latest compiled and minified jQuery -->
  <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-
q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>
  <!-- Latest compiled and minified JavaScript -->
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"
integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy"
crossorigin="anonymous"></script>
</html>
```

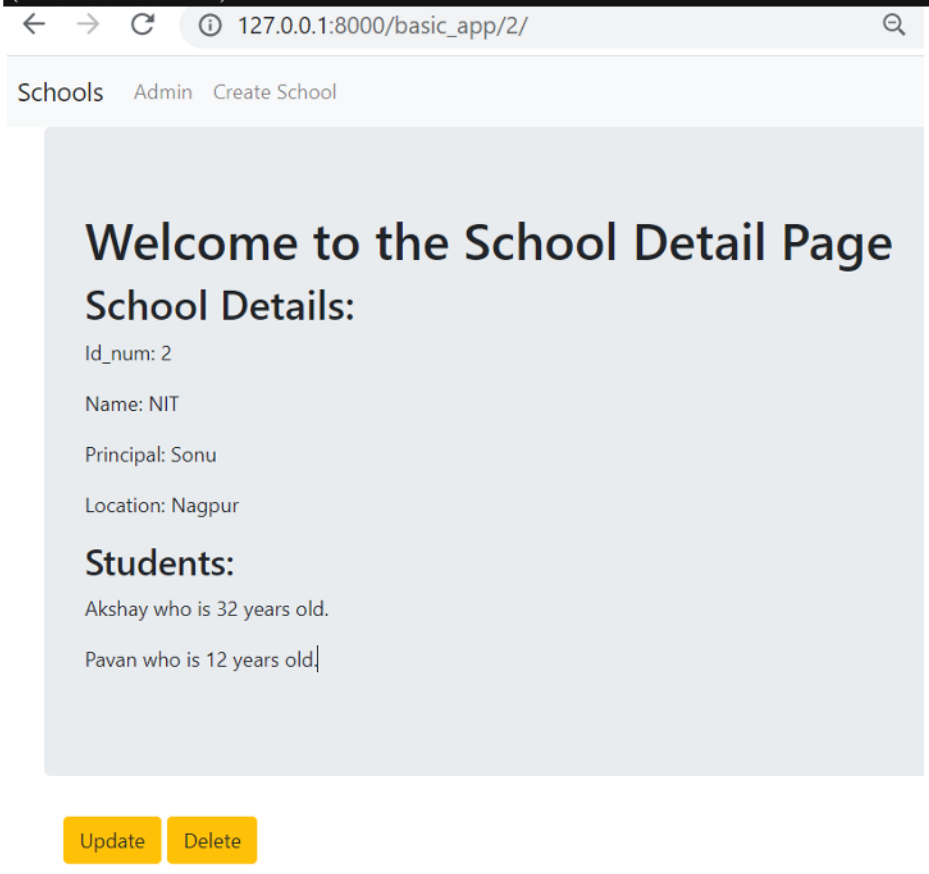
school_list.html

```
{% extends "basic_app/basic_app_base.html" %}
{% block body_block %}
  <div class="jumbotron">

    <h1>Welcome to the List of Schools Page!</h1>
      <ol>
        {% for school in school_list %}
          <h2><li><a href="{{school.id}}/">{{school.name}} </a></li></h2>
        {% endfor %}
      </ol>

  </div>

{% endblock %}
```



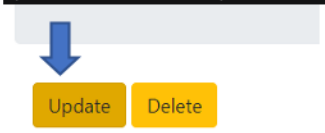
School_details.html

```
{% extends "basic_app/basic_app_base.html" %}
{% block body_block %}
  <div class="jumbotron">
    <h1>Welcome to the School Detail Page</h1>
    <h2>School Details:</h2>
    <p>Id_num: {{school_details.id}}</p>
    <p>Name: {{school_details.name}}</p>
    <p>Principal: {{school_details.principal}}</p>
    <p>Location: {{school_details.location}}</p>
    <h3>Students:</h3>

      {% for student in school_details.students.all %}
        <p>{{student.name}} who is {{student.age}} years old.</p>
      {% endfor %}

  </div>
  <div class="container">
    <a class='btn btn-warning' href="{% url 'basic_app:update' pk=school_details.pk
%}">Update</a>
    <a class='btn btn-warning' href="{% url 'basic_app:delete' pk=school_details.pk
%}">Delete</a>

  </div>
{% endblock %}
```

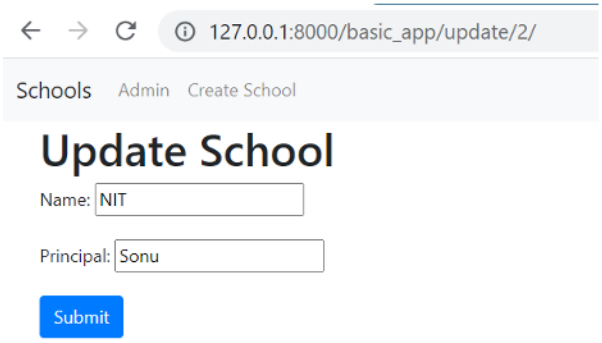


school_form.html (for update button or for create new entry)

```
{% extends "basic_app/basic_app_base.html" %}

{% block body_block %}
<h1>
  {% if not form.instance.pk %}
  Create School
  {% else %}
  Update School
  {% endif %}
</h1>
  <form method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" class='btn btn-primary' value="Submit">

  </form>

{% endblock %}
```



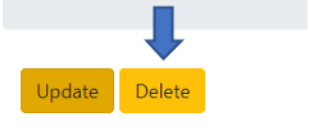
v



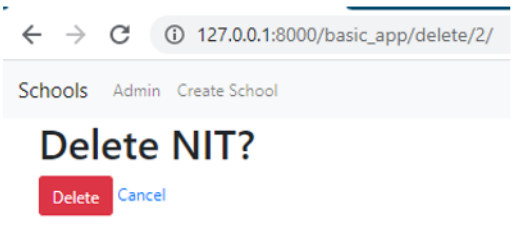
only name and principal fields have permission to update
(def SchoolUpdateView(UpdateView))

School_confirm_delete.html
(after clicking on delete button following html executed i.e. SchoolDeleteView(DeleteView)

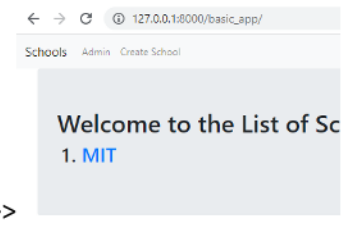
```
{% extends "basic_app/basic_app_base.html" %}

{% block body_block %}
<h1>Delete {{school.name}}?</h1>

<form method="post">
  {% csrf_token %}
  <input type="submit" class="btn btn-danger" value="Delete">
  <a href="{% url 'basic_app:detail' pk=school.pk %} ">Cancel</a>

</form>

{% endblock %}
```



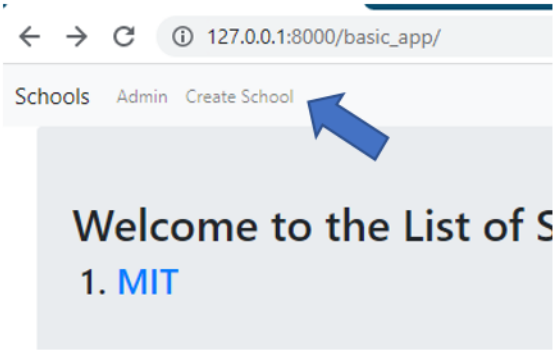
v



>>



after clicking on Delete : NIT was deleted




```
<a class="nav-item nav-link" href="{% url 'basic_app:create' %}">Create School</a>
```

From basic_app_base.html using template tagging CreateView is called with following url

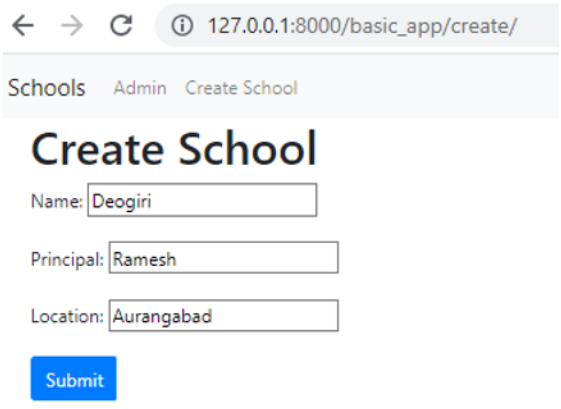
```
path('create/',views.SchoolCreateView.as_view(),name='create'),
```

SchoolCreateView:

```
class SchoolCreateView(CreateView):
    fields = ("name","principal","location")
    model = models.School
```

school_form.html

```
{% extends "basic_app/basic_app_base.html" %}

{% block body_block %}
<h1>
  {% if not form.instance.pk %} #create school
  Create School
  {% else %}
  Update School
  {% endif %}
</h1>
  <form method="POST">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" class='btn btn-primary' value="Submit">

  </form>
{% endblock %}
```



>>>