

PROYECTO INFO COMISION 6

clase 1

- 1) Crear una carpeta vacía “proyecto”
- 2) Crear dentro de la carpeta “proyecto” dos carpetas vacías “entorno” y “repositorio”
- 3) Dentro de la carpeta “entorno” instalar un entorno virtual: **pip install virtualenv**
- 4) Dentro de la carpeta “entorno” creamos un entorno virtual: **python -m venv entornoProyecto**
- 5) activar el entorno: cd \proyecto\entorno\entornoProyecto\Scripts > **activate**
- 6) instalar django: **pip install django**
- 7) creamos nuestra app con django: cd \proyecto\repositorio **django-admin startproject blog**
- 8) como correr el servidor: cd \proyecto\repositorio\blog **python manage.py runserver**
- 9) (dentro de la carpeta blog creamos 3 carpetas) apps | templates | static
- 10) (dentro de la carpeta static creamos 3 carpetas) css | js | img
- 11) (dentro de la carpeta blog/blog creamos 1 carpeta) settings
- 12) (dentro de la carpeta settings) creamos 3 archivos Python: base.py | local.py | production.py
- 13) Agarramos lo que estaba en **settings.py** lo mandamos a **base.py** y borramos el original
- 14) Ctrl + x esto a **local.py**

```
from .base import *

DEBUG = True
ALLOWED_HOSTS = []

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

- 15) En manage.py modificamos esto:

```
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'blog.settings.local')
```

COMANDOS BÁSICOS:

como entrar al entorno virtual (siempre tenemos que utilizarlo para trabajar)

cd significa “*change directory*” o cambio de directorio, **cd [nombreDeCarpeta]** nos permite movernos hacia adelante dentro de nuestras carpetas o para ir hacia atrás es **cd ..**

```
C:\1-INF02023\ proyecto\ repositorio>cd ..  
C:\1-INF02023\ proyecto>cd entorno  
C:\1-INF02023\ proyecto\ entorno>cd entornoProyecto  
C:\1-INF02023\ proyecto\ entorno\ entornoProyecto>cd Scripts  
C:\1-INF02023\ proyecto\ entorno\ entornoProyecto\ Scripts>activate  
(entornoProyecto) C:\1-INF02023\ proyecto\ entorno\ entornoProyecto\ Scripts>
```

como hacer correr nuestro proyecto **python manage.py runserver**

```
(entornoProyecto) C:\1-INF02023\ proyecto\ repositorio\ blog>python manage.py runserver  
Watching for file changes with StatReloader  
Performing system checks...  
  
System check identified no issues (0 silenced).  
June 22, 2023 - 18:24:52  
Django version 4.2.2, using settings 'blog.settings.local'  
Starting development server at http://127.0.0.1:8000/  
Quit the server with CTRL-BREAK.
```

(se sale del mismo con **CTRL + C**)

clase 2

1) en **base.py** configuramos los templates:

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [os.path.join(os.path.dirname(BASE_DIR), 'templates')], ←— añadimos esto  
        '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',  
    ],  
},  
},  
]
```

2) creamos nuestra app usuarios: **django-admin startapp usuarios**

```
(entornoProyecto) C:\1-INF02023\ proyecto\repositorio\blog\apps>django-admin startapp usuarios
```

ATENCIÓN: deben estar dentro del entorno virtual para trabajar y estar parados en la carpeta apps para realizar el comando 😎

3) En la carpeta usuarios, en el archivo **models.py**:

```
from django.db import models  
  
from django.contrib.auth.models import AbstractUser  
  
class Usuario(AbstractUser):  
  
    pass
```

4) En la carpeta usuarios, en el archivo **apps.py**:

```
class UsuarioConfig(AppConfig):  
  
    default_auto_field = 'django.db.models.BigAutoField'  
  
    name = 'apps.usuario'
```

5) en base.py:

```
# SECURITY WARNING: keep the secret key used in production secret!  
  
SECRET_KEY = 'django-insecure-(e8)q0vlxh!u@kgloozu)csmcw*02q4^!z7h@yef6tbovhmd7'  
  
AUTH_USER_MODEL = 'usuarios.Usuario' ←— añadimos esto
```

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'apps.usuarios', ←— añadimos esto
]
```

6) python manage.py makemigrations | python manage.py migrate

```
(entornoProyecto) C:\1-INFO2023\ proyecto\repositorio\blog>python manage.py makemigrations
Migrations for 'usuarios':
  apps\usuarios\migrations\0001_initial.py
    - Create model Usuario

(entornoProyecto) C:\1-INFO2023\ proyecto\repositorio\blog>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, usuarios
```

makemigrations: mira tu proyecto, sus clases y luego mira la base de datos y se fija que diferencias puede haber, makemigrations genera un archivo para migrar esa diferencia, genera un código para impactar eso en la bd

migrate: ejecuta ese código y lo impacta en la bd

- 7) creamos el archivo **views.py** en la carpeta blog
- 8) en este archivo nuevo views añadimos lo siguiente:

```
from django.shortcuts import render

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

- 9) en urls.py en la carpeta blog añadimos lo siguiente:

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

```

urlpatterns = [
    path('admin/', admin.site.urls),
        #1 parametro: es el texto de la url
        #2 parametro: la vista que se va a ejecutar
        #3 parametro: es el nombre de la url (todavia no le damos uso)
    path('', views.Home, name='home'),
]

```

10) en la carpeta **templates** creamos el archivo '**home.html**'

```

<!DOCTYPE html>

<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mi blog</title>
</head>

<body>
    Hola mundo
</body>
</html>

```

11) Después hicimos los mismos tres pasos para la parte de '*Nosotros*'
a) blog/urls.py

```

path('nosotros/', views.Nosotros, name='nosotros'),

```

b) blog/views.py

```

def Nosotros(request):
    return render(request, 'nosotros.html')

```

c) templates/nosotros.html

```

<!DOCTYPE html>

```

```

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Mi blog</title>

</head>

<body>

    Sobre mi !

</body>

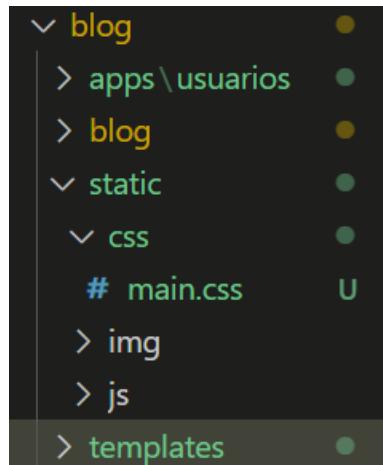
</html>

```

12) añadimos un botón para ir a 'nosotros' en el <body> de home.html:

```
<a href="{% url 'nosotros' %}>IR A NOSOTROS</a>
```

13) Añadimos estilos! en la carpeta css creamos un archivo: main.css



14) dentro de ese archivo main.css agregamos lo siguiente:

```
.contenedor {

    background-color: lightpink;
}
```

15) para que ese estilo impacte sobre nuestro archivo ‘home.html’ lo referenciamos de la siguiente manera:

```
{% load static %} ←— añadimos esto

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" type="text/css" href="{% static 'css/main.css' %}"> ←— añadimos esto

    <title>Mi blog</title>

</head>

<body class="contenedor"> ←— añadimos esto

    Hola mundo

    <a href="{% url 'nosotros' %}">IR A NOSOTROS</a>

</body>

</html>
```

16) en **base.py** en la linea 106 aprox añadimos una configuración para nuestros archivos estáticos:

```
STATIC_URL = '/static/'

STATICFILES_DIRS = (os.path.join(os.path.dirname(BASE_DIR), 'static'),)
```

clase 3

1) Configuración de la base de datos en local.py:

```
DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.mysql',

        'NAME': 'blog_info',

        'USER': 'root',

        'PASSWORD': 'root',
```

```
'HOST': 'localhost',
'PORT': '',
}
}
```

Si alguien tiene un usuario distinto/puerto distinto se puede crear su propio archivo *por ej mica.py* con la diferencia:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'blog_info',
        'USER': 'root',
        'PASSWORD': 'root',
        'HOST': 'localhost',
        'PORT': '1203',
    }
}
```

Y hacer correr los comandos de esta manera por ej: python manage.py runserver--settings=blog.settings.mica

- 2) Bajar la librería que maneja la base de datos: **pip install mysqlclient**
- 3) Creamos la base de datos en workbench con el nombre que definimos (blog_info) y migramos los cambios (makemigrations / migrate)
- 4) Creamos un superusuario: **python manage.py createsuperuser**
- 5) Creamos un archivo html **base** para nuestro blog:

```
{% load static %}

<!DOCTYPE html>

<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" type="text/css" href="{% static 'css/main.css' %}">
    <title>Mi blog | {% block titulo %}{% endblock titulo %}</title>
```

```

</head>

<body class="contenedor">

<div>

<a href="{% url 'home' %}">IR A HOME</a>

<a href="{% url 'nosotros' %}">IR A NOSOTROS</a>

</div>

{% block contenido %}

{% endblock contenido %}

</body>

</html>

```

6) Y hacemos modificaciones correspondientes en los archivos **home** y **nosotros**:

```

{% extends 'base.html' %}

{% load static %}

{% block titulo %}Home{% endblock titulo %}

{% block contenido %}

<h2>Hola mundo</h2>

{% endblock contenido %}

```

7) Agregamos una nueva app: **django-admin startapp noticias**

8) en urls.py de la carpeta blog agregamos su ruta:

```

from django.urls import path, include

#urls de aplicaciones

path('noticias/', include('apps.noticias.urls')),

```

9) en apps.py de noticias modificamos el name:

```

name = 'apps.noticias'

```

Creamos un patrón MVT (Modelo-Vista-Template) para listar noticias:

10) En **urls.py** de la carpeta noticias:

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

app_name = 'noticias'

urlpatterns = [
    path("", views.ListarNoticias, name= 'listar'),
]
```

app_name = 'noticias' ← no se olviden de esto sino les dará el mismo error que me dio en clases :p

11) En **views.py** creamos una vista para listar noticias:

```
from django.shortcuts import render

# Create your views here.

def ListarNoticias(request):
    return render(request, 'noticias/listar.html')
```

12) En templates creamos una carpeta 'noticias' con el archivo **listar.html**:

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

{% load static %}

{% block titulo %}NOTICIAS{% endblock titulo %}

{% block contenido %}
<h2>Aca estan mis noticias</h2>
{% endblock contenido %}
```

13) Añadimos un botón para ir a noticias en **base.html**:

```
<a href="{% url 'noticias:listar' %}">IR A NOTICIAS</a>
```

clase 4

1) Agregamos esta configuracion a base.py:

```

STATIC_URL = '/static/'

STATICFILES_DIRS = (os.path.join(os.path.dirname(BASE_DIR),'static'),)

MEDIA_URL = 'media/'

MEDIA_ROOT = os.path.join(os.path.dirname(BASE_DIR),'media')

```

Y creamos una carpeta ‘media’ a la altura de las carpetas apps/blog/static/templates

En **static** van los archivos estáticos (es decir lo que subo yo) y en **media** los archivos dinámicos (lo que sube el usuario)

2) Creamos modelo noticia modelo categoría (noticias/models.py)

```

from django.db import models

class Categoria(models.Model):
    nombre = models.CharField(max_length = 30)

    def __str__(self):
        return self.nombre

class Noticia(models.Model):
    titulo = models.CharField(max_length = 50)
    resumen = models.CharField(max_length = 100, null = True)
    contenido = models.TextField()
    fecha_publicacion = models.DateTimeField(auto_now_add=True)
    #imagen requiere la libreria pillow
    imagen = models.ImageField(upload_to = 'noticias')

    categoria_noticia = models.ForeignKey(Categoria, on_delete = models.CASCADE) #la otra
    opcion a CASCADE = SET_NULL

    def __str__(self):
        return self.titulo

```

3) pip install pillow

- 4) hacemos las migraciones
- 5) en noticias/admin.py:

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

6) en views.py de noticias

```
from django.shortcuts import render
from .models import Noticia

# Create your views here.

def ListarNoticias(request):
    contexto = {}
    n = Noticia.objects.all()
    contexto['noticias'] = n
    return render(request, 'noticias/listar.html', contexto)
```

7) en templates/noticias/listar.html:

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

{% load static %}

{% block titulo %}Noticias{% endblock titulo %}

{% block contenido %}

<h2>Aca estan mis noticias</h2>

{% for noti in noticias %}
```

```



<p>TITULO: {{noti.titulo}}</p>

<p>FECHA: {{noti.fecha_publicacion}}</p>

<p>CATEGORIA: {{noti.categoría_noticia}}</p>

<p>CUERPO: {{noti.contenido}}</p>

<br>

{% endfor %}

{% endblock contenido %}

```

8) en urls.py de blog

```

from django.contrib import admin

from django.urls import path, include

from . import views

from django.conf import settings

from django.conf.urls.static import static


urlpatterns = [
    path('admin/', admin.site.urls),
    path("", views.Home, name= 'home'),
    path('nosotros/', views.Nosotros, name='nosotros'),


    #urls de aplicaciones
    path('noticias/', include('apps.noticias.urls')),

] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

```

Creamos el MVT (modelo-vista-template) para las noticias individualmente:

9) en urls.py de noticias:

```

from django.urls import path

from . import views


app_name = 'noticias'

```

```
urlpatterns = [
    path("", views.ListarNoticias, name="listar"),
    path('detalle/<int:pk>', views.DetalleNoticias, name='detalle'),
]
```

10) en **views.py** de noticias:

```
def DetalleNoticias(request, pk):
    contexto = {}

    n = Noticia.objects.get(pk = pk) #retorna un objeto
    contexto['noticia'] = n

    return render(request, 'noticias/detalle.html', contexto)
```

11) en templates/noticias/detalle.html

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

{% load static %}

{% block titulo %}Noticia{% endblock titulo %}

{% block contenido %}

<h2>Mi noticia detallada</h2>



<p>TITULO: {{noticia.titulo}}</p>

<p>FECHA: {{noticia.fecha_publicacion}}</p>

<p>CATEGORIA: {{noticia.categoría_noticia}}</p>

<p>CUERPO: {{noticia.contenido}}</p>

<br>

<a href="{% url 'noticias:listar' %}">Volver</a> ← añadí esto post clases es un botón para volver :)

{% endblock contenido %}
```

Si quieren probar las nuevas funciones que añadimos pueden crear noticias desde el sitio de administrador de django y ver como aparecen en el blog 😊 vamos puliendo de a poquito este proyecto, sigan practicando 🎨

clase 5

- 1) Creacion de login/logout: en usuarios/views.py creamos la vista:

```
from django.contrib.auth import authenticate, login, logout
from django.shortcuts import render, redirect
from django.contrib import messages

def user_login(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect('home') # Redirección a la página de home post login
        else:
            messages.error(request, 'Usuario o contraseña invalido, intente de nuevo.')
    return render(request, 'usuarios/login.html')

def user_logout(request):
    logout(request)
    return redirect('login') # Redirección a la página de login post logout
```

- 2) en usuarios/urls.py:

```
from django.urls import path
from .views import user_login, user_logout

urlpatterns = [
```

```
path('login/', user_login, name='login'),  
path('logout/', user_logout, name='logout'),  
]
```

3) En usuarios/login.html:

```
{% extends 'base.html' %}  
  
{% block titulo %}Login{% endblock titulo %}  
  
{% block contenido %}  
  
    <form method="POST">{% csrf_token %}  
  
        {% if messages %}  
  
            {% for message in messages %}  
  
                <div class="alert alert-danger">{{ message }}</div>  
  
            {% endfor %}  
  
        {% endif %}  
  
        <input type="text" name="username" placeholder="Username">  
  
        <input type="password" name="password" placeholder="Password">  
  
        <button type="submit">Login</button>  
  
    </form>  
  
{% endblock %}
```

4) en las urls de blog:

```
path('usuario/', include('apps.usuarios.urls')),
```

5) Agregamos los botones de login y logout:

```
<div>  
  
      
  
    <a href="{% url 'home' %}">IR A HOME</a>  
  
    <a href="{% url 'nosotros' %}">IR A NOSOTROS</a>  
  
    <a href="{% url 'noticias:listar' %}">IR A NOTICIAS</a>  
  
  
    {% if user.is_authenticated %}  
  
        <strong>Hola {{ user.username }}</strong>
```

```

<a href="{% url 'logout' %}">LOGOUT</a>

{% else %}

<a href="{% url 'login' %}">LOGIN</a>

{% endif %}

</div>

```

6) Creación de registro de usuarios, en usuarios/views.py:

```

from django.shortcuts import render

from django.views.generic import CreateView

from django.urls import reverse_lazy

from .forms import RegistroForm

# Create your views here.

class Registro(CreateView):

    #FORMULARIO DJANGO

    form_class = RegistroForm

    success_url = reverse_lazy('login')

    template_name = 'usuarios/registro.html'

```

7) En usuarios/forms.py:

```

from django import forms

from django.contrib.auth.forms import UserCreationForm

from .models import Usuario

class RegistroForm(UserCreationForm):

    email = forms.EmailField(label='Correo', required=True)

```

```
first_name = forms.CharField(label='Nombre', required=True)

last_name = forms.CharField(label='Apellido', required=True)

password1 = forms.CharField(
    label='Contraseña', widget=forms.PasswordInput, required=True)

password2 = forms.CharField(
    label='Confirmar Contraseña', widget=forms.PasswordInput, required=True)

class Meta:
    model = Usuario

    fields = [
        'first_name',
        'last_name',
        'username',
        'email',
        'password1',
        'password2'
    ]
```

8) en usuarios/registro.html

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

{% load static %}

{% block title %}Registro{% endblock %}

{% block contenido %}
```

```
<div>

<h1>Registarse</h1>

<div><b>Para usar la plataforma debe Registrarse</b> </div>

</div>

<form method="post">{%csrf_token%}

{{form.errors}}


<div class="formulario-registro">

<label for="Nombre">Nombre</label>

<input type="text" name="first_name" placeholder="Nombre" required>




<label for="Apellido">Apellido</label>

<input type="text" name="last_name" placeholder="Apellido" required>




<label for="Nombre de Usuario">Nombre De Usuario</label>

<input type="text" name="username" placeholder="Nombre de Usuario" required>




<label for="Correo Electronico">Correo Electronico</label>

<input type="text" name="email" placeholder="correo@gmail.com" required>




<label for="Contraseña">Contraseña</label>

<input type="password" name="password1" placeholder="Contraseña" required>




<label for="Confirmar Contraseña">Confirmar Contraseña</label>

<input type="password" name="password2" placeholder="Confirmar Contraseña" required>
```

```
</div>

<input type="submit" value="registrarse">

</form>

{% endblock contenido %}
```

9) en usuarios/urls.py agregamos la ruta para registrar usuarios

```
from django.urls import path

from . import views

app_name = 'usuarios'

urlpatterns = [
    path('registro/', views.Registro.as_view(), name = 'registro'),
]
```

10) Agregamos un botón para ir a la parte de registro:

```
<div>
    
    <a href="{% url 'home' %}">IR A HOME</a>
    <a href="{% url 'nosotros' %}">IR A NOSOTROS</a>
```

```

<a href="{% url 'noticias:listar' %}">IR A NOTICIAS</a>

    {% if user.is_authenticated %}

        <strong>Hola {{ user.username }}</strong>

        <a href="{% url 'logout' %}">LOGOUT</a>

    {% else %}

        <a href="{% url 'login' %}">LOGIN</a>

        <a href="{% url 'registro' %}">REGISTRO</a>

    {% endif %}

</div>

```

vamos avanzando con nuestro proyecto! a no bajar los brazos que estamos cerca del final 😊💖

clase 6

- 1) Creación de un listado por categoría: noticias/views.py

```

def ListarNoticias(request):
    contexto = {} #diccionario
    id_categoria = request.GET.get("id", None)

    if id_categoria:
        n = Noticia.objects.filter(categoría_noticia = id_categoria)
    else:
        n = Noticia.objects.all() #SELECT * FROM Noticias / lista objetos

    contexto['noticias'] = n

    cat = Categoría.objects.all().order_by('nombre') #ordena por nombre
    contexto['categorías'] = cat

```

```
return render(request, 'noticias/listar.html', contexto)
```

2) listar.html:

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

{% load static %}

{% block titulo %}NOTICIAS{% endblock titulo %}

{% block contenido %}

<h2>Aca estan mis noticias</h2>

<div class="submenu">

<a href="{% url 'noticias:listar' %}">TODAS</a>

{% for cat in categorias %}

<a href="?id={{cat.pk}}">{{cat.nombre}}</a>

{% endfor %}

</div>

{% if noticias %} #pregunta si hay noticias, si no las hay muestra "no hay notis"

{% for noti in noticias %}



<p><a href="{% url 'noticias:detalle' noti.pk %}">TITULO: {{noti.titulo}}</a></p>

<p>FECHA: {{noti.fecha_de_publicacion}}</p>

<p>CATEGORIA: {{noti.categoría_noticia}}</p>

<p>CONTENIDO: {{noti.contenido}}</p>

{% endfor %}
```

```
{% else %}

<h2>no hay notis</h2>

{% endif %}

{% endblock contenido %}
```

AÑADIR NOTICIAS

3) views.py

```
def AddNoticia(request):

    form = NoticiaForm(request.POST or None, request.FILES) ##REQUEST FILE PARA LAS
    IMAGENES

    if request.method == 'POST' and form.is_valid():

        form.save()

        return redirect(reverse('home'))

    return render(request, 'noticias/addNoticia.html', {'form': form})
```

4) creamos un archivo forms.py con lo siguiente:

```
from django import forms

from .models import Noticia, Comentario

class NoticiaForm(forms.ModelForm):

    class Meta:
```

```
model = Noticia  
fields = ['titulo', 'resumen', 'contenido', 'imagen', 'categoria_noticia']
```

5) urls.py:

```
from django.urls import path  
  
from . import views  
  
from django.conf import settings  
  
from django.conf.urls.static import static  
  
  
  
app_name = 'noticias'  
  
  
urlpatterns = [  
  
    path("", views.ListarNoticias, name='listar'),  
  
    path('listarNoticia/', views.mostrarNoticia.as_view()), #clase  
  
    path('detalle/<int:pk>', views.DetalleNoticias, name='detalle'),  
  
    path('addNoticia', views.AddNoticia, name='addnoticia')  
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

6) addnoticia.html:

```
{% extends 'base.html' %}  
  
{% load static %}  
  
  
  
{% block titulo %}NOTICIAS{% endblock titulo %}  
  
  
  
{% block contenido %}  
  
<h2>Crea una noticia</h2>  
  
<form action="" method="POST" enctype="multipart/form-data">  
  
    {% csrf_token %}
```

```

{{ form.as_p }}

<button type="submit">Guardar</button>

</form>

{% endblock contenido %}

```

clase 7

MOSTRAR QUE AUTOR CREO LA NOTICIA

1) noticia/models.py:

```

from apps.usuarios.models import Usuario

#NO OLVIDEN IMPORTAR EL USUARIO PARA USARLO COMO AUTOR :)

class Noticia(models.Model):
    titulo = models.CharField(max_length=250)
    resumen = models.CharField(max_length=100)
    contenido = models.TextField()
    fecha_de_publicacion = models.DateTimeField(auto_now_add=True)
    #para imagen debemos instalar pillow
    imagen = models.ImageField(upload_to='noticias')
    categoria_noticia = models.ForeignKey(Categoría, on_delete=models.CASCADE)
    #SET_NULL
    author = models.ForeignKey(Usuario, on_delete=models.CASCADE,
    default=Usuario.objects.get(is_superuser=True).pk)

    # se agregó el campo author que hace referencia al autor de la noticia, el
    # "default=Usuario.objects.get(is_superuser=True).pk" esta ahí para que tome por defecto (de las
    # noticias que ya tenía creadas) como usuario al superusuario, otra manera de hacerlo es
    # admitiendo valores nulos con "default=null"

    ((otra OTRA manera de hacerlo es simplemente borrar todas las noticias que tenías previo al
    agregado de este campo))

    def __str__(self):
        return self.titulo

```

2) detalle.html:

```
  
  
<p>TITULO: {{ noticia.titulo }}</p>  
  
<p>FECHA: {{ noticia.fecha_de_publicacion }}</p>  
  
<p>AUTOR: {{ noticia.author }}</p>  
  
<p>CATEGORIA: {{ noticia.categoría_noticia }}</p>  
  
<p>CUERPO: {{ noticia.contenido }}</p>  
  
<br>
```

CREAR Y Borrar NOTICIAS

3) modificamos el añadir noticias para que tome el autor también y @login_required solo permite que usuarios logueados creen noticias:

```
from django.contrib.auth.decorators import login_required  
  
@login_required  
  
def AddNoticia(request):  
  
    if request.method == 'POST':  
  
        form = NoticiaForm(request.POST, request.FILES) ##REQUEST FILE PARA LAS  
        IMAGENES  
  
        if form.is_valid():  
  
            noticia = form.save(commit=False)  
  
            noticia.author = request.user #autor de la noticia  
  
            noticia.save()  
  
            return redirect('home')  
  
    else:  
  
        form = NoticiaForm()  
  
  
    return render(request, 'noticias/addNoticia.html', {'form': form})
```

4) en views.py modificamos la funcion DetalleNoticia añadiendo esto:

```
#BORRAR NOTICIA

if request.method == 'POST' and 'delete_noticia' in request.POST:
    noticia.delete()
    return redirect('noticias:listar')
```

- 5) en **detalle.html** añadimos una condición de que si el usuario autenticado coincide con el usuario que es autor de la noticia que se habilite un botón para borrar la noticia:

```
<h2>Mi noticia detallada</h2>

{% if user.is_authenticated and noticia.author == user %}

<form method="POST" action="{% url 'noticias:detalle' pk=noticia.pk %}">
    {% csrf_token %}
    <button type="submit" name="delete_noticia">BORRAR</button>
</form>
```

CREAR COMENTARIOS

- 6) noticias/forms.py

```
from .models import Noticia, Comment

class CommentForm(forms.ModelForm):

    class Meta:
        model = Comment
        fields = ['text'] #campos de mi formulario
        exclude = ['author'] #excluimos al autor del formulario de comentario (no lo va a querer
        como campo cuando creemos uno)

    def __init__(self, *args, **kwargs):
        user = kwargs.pop('user', None)
        super(CommentForm, self).__init__(*args, **kwargs)
        if user:
            self.instance.author = user.username
```

```
def __init__(self, *args, **kwargs)
```

El constructor `__init__` se llama cuando se crea una instancia del formulario. El método toma argumentos posicionales `*args` y argumentos de palabras clave `**kwargs`.

```
user = kwargs.pop('user', None)
```

`kwargs`: Es un diccionario que contiene argumentos de palabras clave. Cuando se utiliza `**kwargs` en la declaración de una función o método, significa que se pueden pasar argumentos de palabras clave adicionales y se recogerán en este diccionario.

`.pop('user', None)` Es un método de los diccionarios en Python. La función `pop()` se utiliza para extraer y eliminar un elemento del diccionario.

'user' es la clave del diccionario que se está buscando y tratando de extraer. `None` es el valor que se devuelve si la clave 'user' no se encuentra en el diccionario.

```
super(CommentForm, self).__init__(*args, **kwargs)
```

Esta línea llama al método `__init__` de la clase base de Django (`forms.ModelForm`) para realizar la inicialización básica del formulario. Se asegura de que todas las características y funcionalidades del formulario base se mantengan.

```
if user:
```

```
self.instance.author = user.username
```

Se verifica si se proporcionó un valor para `user` y se establece el atributo `author` del objeto `self.instance`. `self.instance` representa la instancia del modelo `Comment` asociada al formulario. Se asigna el nombre de usuario del usuario proporcionado al atributo `author`.

7) noticias/models.py:

```
class Comment(models.Model):  
    noticia = models.ForeignKey(Noticia, on_delete=models.CASCADE,  
    related_name='comments')  
  
    author = models.CharField(max_length=100)  
  
    text = models.TextField()  
  
    created_at = models.DateTimeField(auto_now_add=True)  
  
  
def __str__(self):  
    return self.text
```

8) noticias/views.py

```
from django.shortcuts import render, redirect, get_object_or_404
from .models import Noticia, Categoria, Comment
from .forms import NoticiaForm, CommentForm
```

```
def DetalleNoticias(request, pk):
    noticia = get_object_or_404(Noticia, pk=pk)
    comments = noticia.comments.all()

#Borrar Noticia
if request.method == 'POST' and 'delete_noticia' in request.POST:
    noticia.delete()
    return redirect('noticias:listar')

# Comentario
if request.method == 'POST' and 'add_comment' in request.POST:
    form = CommentForm(request.POST)
    if form.is_valid():
        comment = form.save(commit=False)
        comment.noticia = noticia
        comment.author = request.user
        comment.save()
        return redirect('noticias:detalle', pk=pk)
    else:
        form = CommentForm()

context = {
    'noticia': noticia,
    'comments': comments,
```

```
'form': form,  
}  
  
return render(request, 'noticias/detalle.html', context)
```

noticia = get_object_or_404(Noticia, pk=pk) Obtiene la noticia correspondiente a la clave primaria pk de la base de datos. Si no se encuentra la noticia, se muestra una página de error 404.

if request.method == 'POST' and 'add_comment' in request.POST:

etc.

else: form = CommentForm()

Se verifica si la solicitud es de tipo POST y si se ha enviado el parámetro 'add_comment' en el formulario. Si se cumple esa condición, se crea una instancia/objeto del formulario CommentForm utilizando los datos enviados en la solicitud. Luego, se valida el formulario. Si es válido, se crea un comentario relacionado con la noticia actual, utilizando los datos del formulario y el usuario que realizó la solicitud (request.user).

return redirect('noticias:detalle', pk=pk)

Se redirige al usuario a la página de detalle de la noticia

9) noticias/views.py

```
# AÑADIR COMENTARIOS  
  
@login_required  
  
def add_comment(request, noticia_id):  
  
    noticia = get_object_or_404(Noticia, id=noticia_id)  
  
    if request.method == 'POST':  
  
        text = request.POST.get('text')  
  
        author = request.user.username  
  
        # creacion de comentario  
  
        Comment.objects.create(noticia=noticia, author=author, text=text)  
  
    return redirect('noticias:detalle', pk=noticia_id)
```

10) detalle.html:

pregunta si el usuario está autenticado para mostrar una sección de añadir comentarios, si no estás logueado te aparece un mensaje de que debes iniciar sesión para comentar.

```

{% if user.is_authenticated %}

<h3>Añadir Comentario</h3>

<form method="POST" action="{% url 'noticias:add_comment' noticia.pk %}">

    {% csrf_token %}

    {{ form.as_p }}

    <button type="submit">Enviar</button>

</form>

{% else %}

<p>Debe iniciar sesión para añadir un comentario</p>

<a href="{% url 'login' %}">LOGIN</a>

{% endif %}

```

LISTAR / BORRAR COMENTARIOS

11) noticias/views.py:

se fija si: 1. estas logueado 2. tu usuario es el autor de la noticia para borrarla

```

@login_required

def delete_comment(request, comment_id):
    comment = get_object_or_404(Comment, id=comment_id)

    if comment.author == request.user.username:
        comment.delete()

    return redirect('noticias:detalle', pk=comment.noticia.pk)

```

12) detalle.html:

```

<h3>Comentarios</h3>

{% for comment in comments %}

<p>{{ comment.author }} - {{ comment.created_at }}</p>
<p>{{ comment.text }}</p>

<form method="POST" action="{% url 'noticias:delete_comment' comment.id %}">

```

```

    {% csrf_token %}

    {% if comment.author == user.username %}

        <button type="submit">Borrar comentario</button>

    {% endif %}

    </form>

    <hr>

    {% empty %}

    <p>No hay comentarios.</p>

    {% endfor %}

```

muestra el autor, fecha de creación y texto del comentario y si sos el autor del mismo {% if comment.author == user.username %} se te habilita un botón para borrar el comentario

13) noticias/urls.py:

```

path('comment/delete/<int:comment_id>', views.delete_comment, name='delete_comment'),

path('comment/add/<int:noticia_id>', views.add_comment, name='add_comment'),

```

Vamos a ir añadiendo funcionalidades nuevas a nuestro blog. Investigan qué otras cosas puedo añadir a mi blog ahora que saben las temáticas que les tocaron, empezamos a full con el proyecto final! Espero blogs bonitos y funcionales 😊 Ustedes pueden! 💪

clase 8

EDITAR NOTICIAS

1) noticias/views.py

```

from django.http import HttpResponseRedirect

def EditNoticia(request, pk):
    noticia = get_object_or_404(Noticia, pk=pk)

    if request.method == 'POST':
        form = NoticiaForm(request.POST, instance=noticia)
        if form.is_valid():
            noticia = form.save()
            return HttpResponseRedirect(reverse('noticias:detail', args=[noticia.id]))
    else:
        form = NoticiaForm(instance=noticia)

    return render(request, 'noticias/editar.html', {'form': form})

```

```

# Solo el autor puede editar la noticia

if noticia.author != request.user:
    return HttpResponseRedirect("No tenes permiso para editar esta noticia.")


if request.method == 'POST':
    form = NoticiaForm(request.POST, request.FILES, instance=noticia)

    if form.is_valid():
        form.save()
        return redirect('noticias:detalle', pk=pk)

    else:
        form = NoticiaForm(instance=noticia)

    context = {
        'form': form,
    }

    return render(request, 'noticias/editNoticia.html', context)

```

get_object_or_404: para obtener la noticia por su clave primaria (pk), si no se encuentra la noticia, se muestra una página de error 404

Luego verificamos si el usuario actual (el que esta loggeado) es el autor de la noticia, si no coinciden se devuelve un mensaje de error 403 (HttpResponseForbidden) indicando que no tiene permiso para editar la noticia / para usar esta funcionalidad importamos HttpResponseForbidden de django.contrib

```
form = NoticiaForm(request.POST, request.FILES, instance=noticia)
```

Se crea una instancia de “NoticiaForm” utilizando los datos que recibió en la solicitud POST y los archivos adjuntos, la instancia se vincula a la noticia existente que se esta editando

Se verifica si el formulario es valido con form.is.valid() y si lo es se guarda con form.save()

```
return redirect('noticias:detalle', pk=pk)
```

Finalmente, se redirige al usuario a la página de la noticia recién detallada, se pasa el valor de la clave primaria (pk) para identificar la noticia específica

Se pasa la página **noticias/editNoticia.html** para renderizar la página de edición de la noticia

2) templates/noticias/editNoticia.html

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

{% load static %}

{% block titulo %}Editar Noticia{% endblock titulo %}

{% block contenido %}

<h2>Edita la noticia</h2>

<form action="" method="POST" enctype="multipart/form-data">

    {% csrf_token %}

    {{ form.as_p }}

    <button type="submit">Guardar cambios</button>

</form>

{% endblock contenido %}
```

3) noticias/urls.py:

```
path('noticias/<int:pk>/edit/', views.EditNoticia, name='edit_noticia'),
```

<int:pk> indica un número entero y pk representa la clave primaria de la noti, y el name='edit_noticia' es para referenciarla en el html (en este caso la referenciamos por este nombre en detalle.html de esta manera: noticias:edit_noticia)

4) templates/noticias/detalle.html

```
<h2>Mi noticia detallada</h2>

{% if user.is_authenticated and noticia.author == user %}

<a href="{% url 'noticias:edit_noticia' pk=noticia.pk %}">Editar</a>

<form method="POST" action="{% url 'noticias:detalle' pk=noticia.pk %}">
```

```

    {% csrf_token %}

    <button type="submit" name="delete_noticia">BORRAR</button>

</form>

{% endif %}

```

EDITAR COMENTARIOS

1) noticias/views.py

```

from django.contrib import messages

# EDITAR COMENTARIOS

@login_required

def edit_comment(request, comment_id):

    comment = get_object_or_404(Comment, id=comment_id)

    #mensaje de error si no sos el autor

    if comment.author != request.user.username:
        messages.error(request, 'No tienes permisos para editar este comentario.')
        return redirect('noticias:detalle', pk=comment.noticia.pk)

    if request.method == 'POST':

        form = CommentForm(request.POST, instance=comment)

        if form.is_valid():

            form.save()

            return redirect('noticias:detalle', pk=comment.noticia.pk)

    else:

        form = CommentForm(instance=comment)

    context = {
        'form': form,
    }

```

```
'comment': comment,  
}  
  
return render(request, 'noticias/edit_comment.html', context)
```

la función **edit_comment** toma dos parámetros: request (la solicitud) y comment_id (el id del comentario que se va a editar)

get_object_or_404(): trae el comentario basado en el id que proporcionamos o un error 404

Si el autor del comentario no coincide con el nombre de usuario del que realiza la petición muestra un mensaje de error usando messages.error() (para utilizar esta función debemos importarla de django.contrib) y se redirige a la pagina de la noticia donde esta alojada el comentario utilizando redirect()

todo se renderiza en noticias/edit_comment.html

2) templates/noticias/edit_comment.html

```
{% extends 'base.html' %}  
  
{% block contenido %}  
  
<h2>Editar Comentario</h2>  
  
<form method="POST">  
  
    {% csrf_token %}  
  
    {{ form.as_p }}  
  
    <button type="submit">Guardar</button>  
  
</form>  
  
{% endblock contenido %}
```

3) noticias/urls.py:

```
path('comment/edit/<int:comment_id>', views.edit_comment, name='edit_comment'),
```

4) detalle.html:

```
<h3>Comentarios</h3>  
  
{% for comment in comments %}  
  
<p>{{ comment.author }} - {{ comment.created_at }}</p>  
  
<p>{{ comment.text }}</p>  
  
<form method="POST" action="{% url 'noticias:delete_comment' comment.id %}">  
  
    {% csrf_token %}
```

```

{%- if comment.author == user.username %}

    <button type="submit">Borrar comentario</button>

    <a href="{% url 'noticias:edit_comment' comment_id=comment.id %}">Editar
comentario</a>

{%- endif %}

</form>

<hr>

{%- empty %}

<p>No hay comentarios.</p>

{%- endfor %}

```

LISTAR POR CATEGORÍA / POR ANTIGÜEDAD / POR ORDEN ALFABÉTICO

1) noticias/views.py

```

def ListarNoticias(request):
    noticias = Noticia.objects.all()

    # Filtrar por categoría
    categoria = request.GET.get('categoria')
    if categoria:
        noticias = noticias.filter(categoria_noticia=categoria) #filtramos las noticias

    # Filtrar por antigüedad ascendente
    antiguedad_asc = request.GET.get('antiguedad_asc')
    if antiguedad_asc:
        noticias = noticias.order_by('fecha_de_publicacion') #ordenamos las noticias

    # Filtrar por antigüedad descendente
    antiguedad_desc = request.GET.get('antiguedad_desc')
    if antiguedad_desc:

```

```

noticias = noticias.order_by('-fecha_de_publicacion')

# Filtrar por orden alfabético ascendente

orden_asc = request.GET.get('orden_asc')

if orden_asc:

    noticias = noticias.order_by('titulo')

# Filtrar por orden alfabético descendente

orden_desc = request.GET.get('orden_desc')

if orden_desc:

    noticias = noticias.order_by('-titulo')

context = {

    'noticias': noticias,

    'categorias': Categoria.objects.all(), # Asegúrate de tener el modelo 'Categoria' importado y
definido

}

return render(request, 'noticias/listar.html', context)

```

2) detalle.html

```

<h2>Aca estan mis noticias</h2>

<div class="submenu">

<a href="{% url 'noticias:listar' %}">TODAS</a>

{% for cat in categorias %}

<a href="?categoria={{ cat.pk }}">{{ cat.nombre }}</a>

{% endfor %}

<a href="?antiguedad_asc=1">Más antiguas primero</a>

<a href="?antiguedad_desc=1">Más recientes primero</a>

<a href="?orden_asc=1">Orden alfabético ascendente</a>

```

```
<a href="?orden_desc=1">Orden alfabético descendente</a>
```

```
</div>
```

"?antiguedad_asc=1" : el signo de interrogación se utiliza para indicar el parámetro de consulta, estos parámetros se especifican como 'clave=valor', en este caso particular como valor se indica 1, pero podría ser cualquier valor numérico / cadena de caracteres.

En el caso de categoría, este valor es su pk "?categoria={{ cat.pk }}"

esto en la web se ve de la siguiente manera:

① 127.0.0.1:8000/noticias/?antiguedad_asc=1

① 127.0.0.1:8000/noticias/?categoria=2

(el dos después de categoria representando el id de la misma)

clase 10

DISTINTOS USUARIOS PARA EL BLOG

1) usuarios/models.py

```
from django.db import models
from django.contrib.auth.models import AbstractUser
from django.db.models.signals import post_save
from django.dispatch import receiver

class Usuario(AbstractUser):
    #Imagen de perfil de usuario
    imagen = models.ImageField(upload_to='usuarios', default='default-user.png')
```

USUARIOS QUE TENEMOS EN NUESTRO BLOG Y SUS PERMISOS

USUARIO_COLABORADOR = 'Colaborador'

USUARIO_VISITANTE = 'Visitante'

USUARIO_MIEMBRO = 'Miembro'

```
USUARIO_SUPER = 'Superusuario'
```

```
TIPOS_DE_USUARIO = [
    (USUARIO_COLABORADOR, 'Colaborador'), #is_superuser=False, is_staff=True
    (USUARIO_VISITANTE, 'Visitante'), #no es nada / no aparece en la base de datos
    (USUARIO_MIEMBRO, 'Miembro'), #is_superuser=False, is_staff=False
    (USUARIO_SUPER, 'Superusuario'), #is_superuser=True, is_staff=True
]
```

#atributo que determina el tipo de usuario, por default es usuario_miembro (cuando te registras en el blog toma este)

```
tipo_usuario = models.CharField(max_length=20, choices=TIPOS_DE_USUARIO,
default=USUARIO_MIEMBRO)
```

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

Señal para asignar el tipo de usuario "Superusuario" cuando se crea un superusuario

```
@receiver(post_save, sender=Usuario)
def asignar_tipo_usuario(sender, instance, created, **kwargs):
    if created and instance.is_superuser:
        instance.tipo_usuario = Usuario.USUARIO_SUPER
        instance.save()
        print(f'Usuario {instance.username} es un Superusuario.')
```

Señal para asignar el tipo de usuario "Miembro" cuando se crea un usuario

```
@receiver(post_save, sender=Usuario)
def asignar_miembro(sender, instance, created, **kwargs):
    if created and not instance.is_superuser:
        instance.tipo_usuario = Usuario.USUARIO_MIEMBRO
        instance.save()
```

```
print(f"Usuario {instance.username} es un Miembro.")
```

con class Usuario(AbstractUser) definimos nuestra clase para usuario que hereda de 'AbstractUser', una clase de usuario de Django que se encarga de la autenticación y manejo de usuarios.

Se agregó un ImageField para que los usuarios puedan tener una foto de perfil, la imagen se almacena en la carpeta 'usuarios' en la carpeta 'media', si el usuario no proporciona ninguna imagen se toma una por default (default-user.png)

Se definen constantes para los diferentes tipos de usuarios que tendrá el blog, como "Colaborador", "Visitante", "Miembro" y "Superusuario" y se crea una lista de tuplas, cada tupla contiene dos elementos: el valor que se almacenará en la base de datos y la etiqueta que se mostrará al usuario en el formulario.

SEÑALES:

se definen dos señales post_save (se dispara después de que se guarda una instancia de Usuario) ambas señales están conectadas a dos funciones, una te asigna el tipo de usuario como Superusuario en la base de datos y la otra como Miembro

más info sobre señales en django:

<https://docs.hektorprofe.net/django/web-playground/introduccion-senales-signals/>

2) usuarios/forms.py

```
from django import forms
from django.contrib.auth.forms import UserCreationForm
from .models import Usuario

class RegistroForm(UserCreationForm):
    email = forms.EmailField(label='Correo', required=True)
    first_name = forms.CharField(label='Nombre', required=True)
    last_name = forms.CharField(label='Apellido', required=True)
    username = forms.CharField(label='Nombre de usuario', required=True)
    password1 = forms.CharField(label='Contraseña', widget=forms.PasswordInput, required=True)
    password2 = forms.CharField(label='Confirmar Contraseña', widget=forms.PasswordInput, required=True)
    imagen = forms.ImageField(label='Imagen de perfil', required=False)
```

```
class Meta:  
    model = Usuario  
    fields = [  
        'first_name',  
        'last_name',  
        'username',  
        'email',  
        'password1',  
        'password2',  
        'imagen',  
    ]
```

3) usuarios/views.py

```
# Vista para el registro de usuarios  
  
class Registro(CreateView):  
    form_class = RegistroForm  
    success_url = reverse_lazy('login')  
    template_name = 'usuarios/registro.html'
```

4) usuarios/admin.py

```
from django.contrib import admin  
  
from django.contrib.auth.admin import UserAdmin  
  
from .models import Usuario  
  
  
# Agregar el modelo Usuario personalizado al panel de administración  
  
class UsuarioAdmin(UserAdmin):  
    model = Usuario  
    list_display = ['username', 'email', 'first_name', 'last_name', 'is_staff', 'tipo_usuario']
```

```
fieldsets = UserAdmin.fieldsets + (  
    (None, {'fields': ('imagen', 'tipo_usuario')}),  
)  
  
admin.site.register(Usuario, UsuarioAdmin)
```

Creamos una clase UsuarioAdmin que hereda de UserAdmin, que es el panel de administración predeterminado para el modelo de usuario de Django.

La variable **list_display** se utiliza para indicar qué campos del modelo Usuario se mostrarán en la lista de usuarios en el panel de administración.

La variable **fieldsets** se utiliza para definir los conjuntos de campos que se muestran en el formulario de edición del usuario en el panel de administración. En este caso, se agrega un conjunto de campos adicional que contiene la imagen de perfil (imagen) y el tipo de usuario (tipo_usuario).

tipo_usuario está presente tanto en **list_display** como en **fieldsets** para mostrarlo como una columna en la lista de usuarios y permitir su edición en el formulario de edición del usuario en el panel de administración.

5) usuarios/urls.py

```
from django.urls import path  
  
from .views import user_login, user_logout  
  
from . import views  
  
from django.conf import settings  
from django.conf.urls.static import static  
  
  
urlpatterns = [  
    path('login/', user_login, name='login'),  
    path('logout', user_logout, name='logout'),  
    path('registro/', views.Registro.as_view(), name='registro')  
  
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

6) templates/usuarios/registro.html

```

{% extends 'base.html' %}

{% load static %}

{% block title %}Registro{% endblock %}

{% block contenido %}

<div>

    <h1>Regístrate</h1>

    <div><b>Para usar la plataforma debes registrarte</b> </div>

</div>

<form method="post" enctype="multipart/form-data">

    {% csrf_token %}

    {{ form.as_p }}

    <input type="submit" value="Registrarse">

</form>

{% endblock contenido %}

```

7) templates/noticias/detalle.html

```

{% extends 'base.html' %}

{% load static %}

{% block contenido %}

    <h2>Mi noticia detallada</h2>

    {% if user.is_authenticated and noticia.author == user or user.is_staff or user.is_superuser %}

        <a href="{% url 'noticias:edit_noticia' pk=noticia.pk %}">Editar</a>

    {% endif %}

    <form method="POST" action="{% url 'noticias:detalle' pk=noticia.pk %}">

```

```

    {% csrf_token %}

    <button type="submit" name="delete_noticia">BORRAR</button>

</form>

{% endif %}



<p>TITULO: {{ noticia.titulo }}</p>

<p>FECHA: {{ noticia.fecha_de_publicacion }}</p>

<p>AUTOR: {{ noticia.author }}</p>

<p>CATEGORIA: {{ noticia.categoría_noticia }}</p>

<p>CUERPO: {{ noticia.contenido }}</p>

<br>

<h3>Comentarios</h3>

{% for comment in comments %}

    <p></img>{{comment.author}}</p>

    <p>{{ comment.text }}</p>

    <p>{{ comment.created_at }}</p>

    <form method="POST" action="{% url 'noticias:delete_comment' comment.id %}">

        {% csrf_token %}

        {% if comment.author == user.username or user.is_staff or user.is_superuser %}

            <button type="submit">Borrar comentario</button>

            <a href="{% url 'noticias:edit_comment' comment_id=comment.id %}">Editar comentario</a>

        {% endif %}

    </form>

    <hr>

    {% empty %}

    <p>No hay comentarios.</p>

{% endfor %}

```

```
{% if user.is_authenticated %}

<h3>Añadir Comentario</h3>

<form method="POST" action="{% url 'noticias:add_comment' noticia.pk %}">

    {% csrf_token %}

    {{ form.as_p }}

    <button type="submit">Enviar</button>

</form>

{% else %}

<p>Debe iniciar sesión para añadir un comentario</p>

<a href="{% url 'login' %}">LOGIN</a>

{% endif %}

<br><br>

<a href="{% url 'noticias:listar' %}">VOLVER</a>

{% endblock contenido %}
```

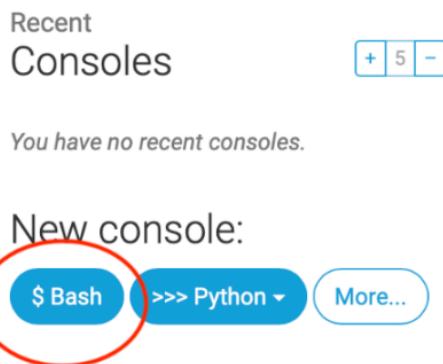
clase 11

DEPLOY PYTHONANYWHERE

- 1) creamos una archivo de texto con las dependencias de nuestro proyecto: el mismo se lo hace con **pip freeze > requirements.txt**

```
(entornoProyecto) PS C:\1-INFO2023\proyecto\repositorio\blog> pip freeze > requirements.txt
```

- 2) Creamos una cuenta en python anywhere:
<https://www.pythonanywhere.com/registration/register/beginner/>
- 3) Abri una consola de bash:



- 4) Copia el link del repo en github y clonalo en esa consola: **git clone [link repo]**

```
Bash console 29624206
20:16 ~ $ git clone https://github.com/micaelacalvente/proyectoINFO.git
Cloning into 'proyectoINFO'...
```

En el caso de que te pida tu nombre de usuario y un token de acceso así se genera:
<https://misovirtual.virtual.uniandes.edu.co/codelabs/access-token-github/index.html#0>

- 5) Creamos un entorno virtual:

```
Bash console 29624206
20:21 ~ $ mkvirtualenv --python=/usr/bin/python3.10 entornoProyecto
mkvirtualenv --python=/usr/bin/python3.10 entornoProyecto
```

la versión de python (hasta ahora solo admite hasta la 3.10) | el nombre de tu entorno

- 6) nos movemos hasta la carpeta donde tenemos nuestro manage.py con el comando cd
- 7) instalamos las dependencias: **pip install -r requirements.txt**

```
(entornoProyecto) 20:30 ~/proyectoINFO/blog (main)$ pip install -r requirements.txt
```

- 8) clickeamos la sección ‘WEB’ y creamos nuestra nueva aplicación web seleccionando la opción **add a new web app**:

You have no web apps
To create a PythonAnywhere-hosted web app, click the "Add a new web app" button to the left.

- 9) seleccionamos la configuración manual:

Select a Python Web framework
...or select "Manual configuration" if you want detailed control.

- » Django
- » web2py
- » Flask
- » Bottle
- » **Manual configuration (including virtualenvs)**

What other frameworks should we have here? Send us some feedback using the link at the top of the page!

[Cancel](#) [« Back](#) [Next »](#)

- 10) colocamos la ruta de nuestro entorno virtual:

Virtualenv:

Use a virtualenv to get different versions of flask, django etc from our default system ones. [More info here](#). You need to **Reload your web app** to activate it; NB - will do nothing if the virtualenv does not exist.

/home/micaelainfo/.virtualenvs/entornoProyecto

Start a console in this virtualenv

11) modificamos el archivo wsgi:

Code:

What your site is running.

Source code: [Enter the path to your web app source code](#)
Working directory: [/home/micaelainfo/](#)
WSGI configuration file: [/var/www/micaelainfo_pythonanywhere_com_wsgi.py](#)
Python version: 3.10 

 /var/www/micaelainfo_pythonanywhere_com_wsgi.py

```
1 # ++++++ DJANGO ++++++
2 # To use your own django app use code like this:
3 import os
4 import sys
5 #
6 ## assuming your django settings file is at '/home/micaelainfo/mysite/mysite/settings.py'
7 ## and your manage.py is at '/home/micaelainfo/mysite/manage.py'
8 path = '/home/micaelainfo/proyectoINFO/blog'
9 if path not in sys.path:
10     sys.path.append(path)
11 #
12 os.environ['DJANGO_SETTINGS_MODULE'] = 'blog.settings.production'
13 #
14 ## then:
15 from django.core.wsgi import get_wsgi_application
16 application = get_wsgi_application()
17
```

12) Creamos nuestra base de datos en ‘databases’ y obtenemos las credenciales necesarias para utilizarlas en nuestro archivo de producción:

 pythonanywhere
by ANACONDA

Dashboard Consoles Files Web Tasks Databases

Warning You have not confirmed your email address yet. This means that you will not be able to reset your password if you lose it. If you cannot find your confirmation email anymore, send yourself a new one [here](#).

MySQL MySQL settings Postgres

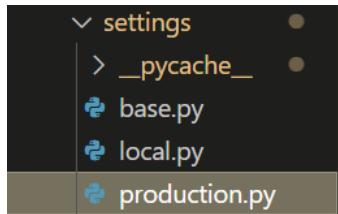
Connecting:
Use these settings in your web applications.

Database host address: micaelainfo.mysql.pythonanywhere-services.com
Username: micaelainfo

Your databases:
Click a database's name to start a MySQL console logged in to it.

Name
micaelainfo\$default

- 13) tenemos que asegurarnos de que tenemos un archivo **production.py** o **prod.py** dentro de nuestro settings:



```
blog > blog > settings > production.py > ...
1   from .base import *
2
3   # SECURITY WARNING: don't run with debug turned on in production!
4
5   # Database
6   # https://docs.djangoproject.com/en/4.2/ref/settings/#databases
7
8   DATABASES = {
9       'default': {
10           #engine = motor de base de datos
11           'ENGINE': 'django.db.backends.mysql',
12           'NAME': 'micaelainfo$default',
13           'USER': 'micaelainfo',
14           'PASSWORD': 'informatorio2023',
15           'HOST': 'micaelainfo.mysql.pythonanywhere-services.com',
16           'PORT': '',
17       }
18 }
```

- 14) en base.py tenemos que agregar la configuración: DEBUG = False y en allowed hosts nuestro sitio:

```
DEBUG = False

ALLOWED_HOSTS = ['micaelainfo.pythonanywhere.com']
```

- 15) Tenes que definir una variable STATIC_ROOT que indica a django donde copiar todos los estáticos para que nuestra web los tome al igual que una para nuestros archivos dinámicos:

```

STATIC_URL = '/static/'
STATICFILES_DIRS = (os.path.join(os.path.dirname(BASE_DIR), 'static'),)

if not DEBUG:
    STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')

MEDIA_URL = 'media/'
MEDIA_ROOT = os.path.join(os.path.dirname(BASE_DIR), 'media')

```

- 16) En manage.py modificamos los settings para que utilize el archivo de producción:

```

def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'blog.settings.production')
    try:

```

IMPORTANTE: Cuando realizas estos cambios en tu compu local tenes que subir los cambios a tu repo (**git add . | git commit -m “mensaje” | git push**) y desde el bash de pythonanywhere hacer un **git pull** para traer los cambios

- 17) indicamos las rutas de ambos archivos (dinamicos y estaticos):

Static files:

Files that aren't dynamically generated by your code, like CSS, JavaScript or uploaded files, can be served much faster straight off the disk if you specify them here. You need to **Reload your web app** to activate any changes you make to the mappings below.

URL	Directory	Delete
/static/	/home/micaelainfo/proyectoINFO/blog/blog/staticfiles	
/media/	/home/micaelainfo/proyectoINFO/blog/media	

- 18) recolectamos los archivos estáticos con collectstatic

```
(entornoProyecto) 21:26 ~/proyectoINFO/blog (main)$ ./manage.py collectstatic
```

- 19) migramos los cambios a la base de datos creada con migrate:

```
(entornoProyecto) 21:26 ~/proyectoINFO/blog (main)$ ./manage.py migrate
```

en el caso de que te aparezca un error al ejecutar este comando, puede ser que no tengas los permisos para ejecutar el mismo, para otorgar permisos a un archivo hacemos uso del comando **chmod**:

```
(entornoProyecto) 21:26 ~/proyectoINFO/blog (main)$ chmod +x manage.py
```

cómo chequear que permisos tiene un archivo: **ls -l**:

```
07:38 ~/proyectoINFO/blog (main)$ ls -l
total 28
drwxrwxr-x 4 micaelainfo registered_users 4096 Jul 27 20:19 apps
drwxrwxr-x 5 micaelainfo registered_users 4096 Jul 27 20:58 blog
-rw-rw-r-- 1 micaelainfo registered_users 0 Jul 27 20:19 db.sqlite3
-rwxrwxr-x 1 micaelainfo registered_users 671 Jul 27 20:52 manage.py
drwxrwxr-x 4 micaelainfo registered_users 4096 Jul 27 20:19 media
-rw-rw-r-- 1 micaelainfo registered_users 166 Jul 27 20:30 requirements.txt
drwxrwxr-x 4 micaelainfo registered_users 4096 Jul 27 20:19 static
drwxrwxr-x 4 micaelainfo registered_users 4096 Jul 27 20:19 templates
```

d: indica que es un directorio (carpeta)
r: read (permiso de lectura)
w: write (permiso de escritura)
x: execute (permiso para ejecutar)

20) al ser una base de datos nueva no tenes datos en la misma, entonces tenes que crear tu superusuario de nuevo:

```
(entornoProyecto) 21:26 ~/proyectoINFO/blog (main)$ python manage.py createsuperuser
```

USO DE LA CONSOLA DE MYSQL EN PYTHONANYWHERE

21) inicializamos la consola clickeando en ‘MySQL’ en la sección Consoles:

Start a new console:

Python: [3.10](#) / [3.9](#) / [3.8](#) / [3.7](#) / [3.6](#) IPython: [3.10](#) / [3.9](#) / [3.8](#) / [3.7](#) / [3.6](#) PyPy: [2](#) / [3](#)
Other: [Bash](#) | [MySQL](#)
Custom: [+](#)

comandos mas usados

22) mostrar las bases de datos disponibles: **show databases;** ← no olvidar los ; para los comandos

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| micaelainfo$default |
| performance_schema |
+-----+
3 rows in set (4.52 sec)
```

23) usar la base de datos: **use [nombre_de_la_bd];**

```
mysql> use micaelainfo$default
Database changed
```

24) mostrar nuestras tablas: **show tables;**

```
mysql> show tables;
+-----+
| Tables_in_micaelainfo$default |
+-----+
| auth_group
| auth_group_permissions
| auth_permission
| django_admin_log
| django_content_type
| django_migrations
| django_session
| noticias_categoria
| noticias_comment
| noticias_noticia
| usuarios_usuario
| usuarios_usuario_groups
| usuarios_usuario_user_permissions |
+-----+
13 rows in set (0.00 sec)
```

25) luego de eso todos los comandos típicos de sql se mantienen por ej select:

```
mysql> select * from noticias_noticia;
+----+----+----+----+
| id | titulo | resumen | contenido | fecha_de_publicacion |
+----+----+----+----+
| 2  | noti   | noti    | noti     | 2023-07-29 06:00:11.358940 |
| 3  | noti   | noti    | noti     | 2023-07-29 06:03:27.144091 |
+----+----+----+----+
2 rows in set (0.02 sec)
```

26) para limpiar la consola el comando es CTRL + L

TIPS FINALES: podes seguir modificando tu proyecto una vez subido a tu servidor, no te olvides que al hacer comandos que requieran del manage.py (runserver / makemigrations / migrate) debemos indicar que queremos usar las configuraciones locales ya que cambiamos la opción por default por la de producción:

```
\blog> python manage.py runserver --settings=blog.settings.local
```

y por supuesto pushear los cambios y traerlos a tu consola de pythonanywhere!

happy coding 😊

comandos básicos de git

Command	Purpose
git config	Set configuration values on the global or local project level
git init	Initialize a git repository
git clone	Copy a git repository from a remote source and set as origin
git status	Check the status of files since the last commit
git add	Move changes to staging area
git commit	Commit changes from the staging area
git push	Push changes from the local repository to remote
git pull	Get latest updates/changes from the remote
git branch	Get a list of all branches
git checkout	Switch to a different branch
git stash	Save changes that will not yet be committed
git merge	Merge two branches
git reset	Set the index to the latest commit
git remote	Check remote/source



@SERGIECODE

GIT CHEAT SHEET

Configurar

Establezca el nombre y el correo electrónico que se adjuntará a sus confirmaciones y etiquetas

```
$ git config --global user.name "Sergie Code"  
$ git config --global user.email "sergiecode@gmail.com"
```

Conceptos básicos de Git

main: rama de desarrollo por defecto

origin: repo por defecto

HEAD: rama actual

HEAD^: padre de HEAD

HEAD~4: tatarabuelo de HEAD

Iniciar un proyecto

Crear un repositorio local

```
$ git init
```

Descargar un repo remoto

```
$ git clone <url>
```

Hacer un cambio (add y commit)

Registrar cambios para próximo commit en todos los archivos:

```
$ git add .
```

Añadir punto de control en git con detalle:

```
$ git commit -m "mensaje de confirmación"
```

Añade al punto de control todos los cambios realizados en los archivos registrados y los confirma

```
$ git commit -am "mensaje de confirmación"
```

Ramas (branch)

Muestra todas las ramas locales. Añade *-r* para mostrar todas las ramas remotas. *-a* para todas las ramas.

```
$ git branch
```

Crear una nueva rama

```
$ git branch <nueva-rama>
```

Cambiar a una rama y actualizar el directorio de trabajo (tiene que existir)

```
$ git checkout <branch>
```

Crear una nueva rama y cambiar a ella

```
$ git checkout -b <newbranch>
```

Eliminar una rama fusionada

```
$ git branch -d <branch>
```

Eliminar una rama, ya sea fusionada o no

```
$ git branch -D <branch>
```

Añadir una etiqueta de versión

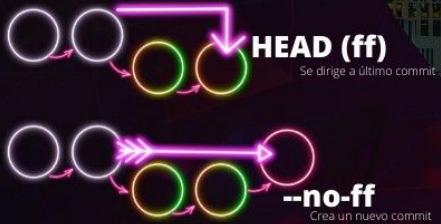
```
$ git tag <nombre de la etiqueta>
```



@SERGIECODE

GIT CHEAT SHEET

Fusión(merge)
Fusionar la rama *a* en la rama *b*. Añadir la opción *--no-ff* para una fusión sin avance rápido



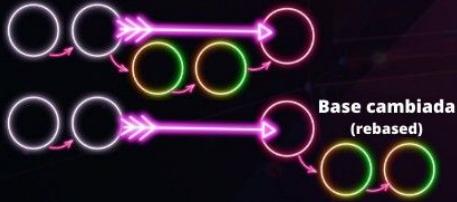
--no-ff
Crea un nuevo commit

```
$ git checkout b
$ git merge a
```

Combinar y aplastar todos los commits en un nuevo commit

```
$ git merge --squash a
```

Cambiar la base(rebase)
Cambiar la base de la rama de la función a la principal (para incorporar los nuevos cambios realizados en la principal). Evita que se realicen confirmaciones innecesarias en la rama principal, manteniendo el historial limpio.



```
$ git checkout feature
$ git rebase main
```

Limpiar interactivamente los commits de una rama antes de cambiar la base a "main"

```
$ git rebase -i main
```

Cambio de base interactivo de los últimos 3 commits en la rama actual

```
$ git rebase -i Head~3
```

Revisar su Repo
Lista de archivos nuevos o modificados que aún no han sido confirmados

```
$ git status
```

Listar el historial de commits, con sus respectivos IDs

```
$ git log --oneline
```

Muestra los cambios en los archivos que no están en fase de desarrollo. Para mostrar los cambios en los archivos en fase, añadir la opción *--cached*

```
$ git diff
```

Muestra los cambios entre dos confirmaciones

```
$ git diff commit1_ID commit2_ID
```



@SERGIECODE

GIT CHEAT SHEET

Deshacer cambios

Mover (y/o renombrar) un archivo y mover la etapa

```
$ git mv <ruta_existente> <ruta_nueva>
```

Eliminar un archivo del directorio de trabajo y del área de preparación(index/stage), y luego escenificar la eliminación

```
$ git rm <archivo>
```

Eliminar sólo del área de preparación(index/stage)

```
$ git rm --cached <fichero>
```

Ver una confirmación anterior (sólo lectura)

```
$ git checkout <id_commit>
```

Crear un nuevo commit, revirtiendo los cambios de un commit especificado

```
$ git revert <identificación_de_comisión>
```

Volver a un commit anterior y borrar todos los commits anteriores a él . Añade la --hard para borrar también los cambios del área de trabajo

```
$ git reset <commit_ID>
```

Almacenamiento

Almacena los cambios modificados y escalonados. Para incluir los archivos no rastreados, añada la bandera -u. Para los archivos no rastreados e ignorados, añada la bandera -a.

```
$ git stash
```

Como el anterior, pero añadiendo un comentario.

```
$ git stash save "comment"
```

Stash parcial. Almacena sólo un archivo, una colección de archivos, o cambios individuales dentro de los archivos

```
$ git stash -p
```

Listar todos los stashes

```
$ git stash list
```

Reaplicar el stash sin borrarlo

```
$ git stash apply
```

Vuelve a aplicar el stash en el índice 2, y luego bórralo de la lista de stash. Omitir stash@{n} para sacar el stash más reciente.

```
$ git stash pop stash@{2}
```

Muestra el resumen diferencial del stash 1. Colocá -p para ver el diferencial completo.

```
$ git stash show stash@{1}
```

Eliminar el stash en el índice 1. Omitir stash@{n} para eliminar el último stash realizado

```
$ git stash drop stash@{1}
```

Borrar todos los stashes

```
$ git stash clear
```



@SERGIECODE

GIT CHEAT SHEET

Sincronización

Añadir un repo remoto

```
$ git remote add <alias> <url>
```

Ver todas las conexiones remotas. Añade la bandera -v para ver las urls.

```
$ git remote
```

Eliminar una conexión

```
$ git remote remove <alias>
```

Renombrar una conexión

```
$ git remote rename <antiguo> <nuevo>
```

Obtener todas las ramas del repositorio remoto (sin fusionar)

```
$ git fetch <alias>
```

Obtener una rama específica

```
$ git fetch <alias> <branch>
```

Obtener la copia del repositorio remoto de la rama actual, y luego fusionar

```
$ git pull
```

Mover (rebase) sus cambios locales sobre los nuevos cambios realizados en el repositorio remoto (para una historia limpia y lineal)

```
$ git pull --rebase <alias>
```

Subir el contenido local al repositorio remoto

```
$ git push <alias>
```

Subir a una rama (luego se puede hacer un pull request)

```
$ git push <alias> <branch>
```