

Contenido

Creación del proyecto en django:	3
Creación de una aplicación dentro del proyecto en django:	3
Configurar Pylint en Visual Studio Code para que me ayude en reconer el código bien	4
Conexión DJANGO - MYSQL:	4
Crear tablas apartir de modelos:	5
Hacer cambios en la estructura de las tablas y de modelos:	5
Guardar registros usando modelos:	6
Guardar registros usando parámetros de la URL :	7
Sacar datos y elementos de la base de datos.....	8
Actualizar registros en la base de datos.....	9
Listar todos los artículos.....	10
Limit y Order By	11
Eliminar registros :	11
Consultas con condiciones filter y lookups :	12
Exclude	13
Ejecutar SQL desde Django	13
OR en consultas con el ORM	14
Recibir datos del formulario por GET	15
Recibir datos del formulario por POST	17
Formularios basados en clases.....	19
Recibir datos y guardar el formulario (Django Form API) :	21
Validar formularios en django (BUSCAR VALIDATOR EN LA DOCUMENTACION DE DJANGO) :	22
Mensajes flash / sesiones flash (BUSCAR MESSAGES EN LA DOCUMENTACION DE DJANGO) :	24
Primeros pasos con el panel de administración de django:	27
Clase META en los modelos de Django 3 (buscar en google meta clase django)	29
Cambiar nombre de las apps.....	30
Metodo mágico para imprimir objetos (modelos), buscar en google (magic methods python)	31
Mostrar campos de solo lectura	32
Subir imágenes en Django.....	32
Mostrar imágenes subidas en la web	34
Cambiar el titulo del panel.....	35
Cambiar el subtítulo del panel (pestaña del navegador).....	35
Menu dinamico y context processors.	36
Mejorar la configuración de rutas.....	38
INVESTIGARLO EN GOOGLE COMO FUNCION include en urls.py.....	38
Usar un editor de texto enriquecido en Django.	39
Modificar el modelo y agregar orden de paginas.	42
Relaciones entre los modelos.	44
Guardar usuario con el inicio de sesion.....	45
Subconsultas en django.	46
Mostrar error 404.....	47

Datos relacionados entre modelos y relaciones inversas.....	48
Personalizar listados.	50
Como hacer una paginación en Django 3.	53
Formulario de registro de usuarios.	55
Mensaje flash en el registro de usuario	57
Login de usuarios	58
Mostrar usuario identificado.....	60
Cerrar sesión y restringir el acceso.....	63
Llenar Choices de manera dinámica apartir de una tabla de la base de datos.....	65
Llenar Choices de manera dinámica apartir de una tabla de la base de datos.....	66
Guardar datos ManytoMany en tablas relacionadas por.	67
Modificar y eliminar datos ManytoMany en tablas relacionadas por.	70
Enviando parámetros con el redirect:	72
Cambio de contraseña con forms :	73
Bibliografia.....	76

Creación del proyecto en django:

- Ubicarse en la carpeta donde tengo los proyectos de django.
- Teclear en el símbolo del sistema: **django-admin startproject nombredelproyecto**
- Teclear en el símbolo del sistema: **python manage.py help** muestra los comandos de ayuda
- Teclear **pip install pymysql** esto es para instalar el cliente de mysql para python.
- Se crea la base de datos en mysql
- Colocar en **settings.py** el siguiente código en la parte de **DATABASES**, sustituir la conexión de sqlite que tiene por esta:

```
• 'ENGINE': 'django.db.backends.mysql',  
•     'NAME': 'dbnombredelabase',  
•     'USER': 'root',  
•     'PASSWORD': 'password',  
•     'HOST': 'localhost',  
•     'PORT': '3306',
```

- Se teclea **python manage.py migrate**
- Y por ultimo **python manage.py runserver**

Creación de una aplicación dentro del proyecto en django:

- En símbolo de sistema ubicarse en la carpeta del proyecto y teclear **python manage.py startapp NOMBREDELAAPP**
- En settings.py agregar la **app** que se creara, en el bloque de código:
-

Configurar Pylint en Visual Studio Code para que me ayude en reconer el código bien

- En símbolo de sistema teclear **pip install pylint-django**.
- En **visual studio code** ir a **File->References->Settings->Python**
- Buscar **pylint** y buscar el encabezado del recuadro que dice **Python-Linting: Pylint Path** y en el recuadro poner: **pylint-django**

Conexión DJANGO - MYSQL:

- Crear la base de datos en **MySQL**
- En el símbolo del sistema **C:\proyectosdjango\ProyectoDjango>** teclear: **pip install pymysql**
- Sustituir la conexión de **sqlite** por **mysql** en el archivo **settings.py** quedando de la siguiente forma:

```
# Database
# https://docs.djangoproject.com/en/2.1/ref/settings/#databases

DATABASES = {
    'default': {
        #'ENGINE': 'django.db.backends.sqlite3',
        #'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'dbproyectodjango',
        'USER': 'root',
        'PASSWORD': '',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

Crear tablas apartir de modelos:

- Despues de crear los campos en **models.py** se tiene que realizar lo siguiente:
- En el símbolo del sistema ubicarse en la carpeta del proyecto.
- Teclear **python manage.py makemigrations**, después.
- Teclear **python manage.py sqlmigrate NOMBRE_DE_LA_APP** y los primeros 4 **numeros de la migration** por ejemplo: **0001**, después.
- Teclear **python manage.py migrate** es ahí donde crea las **tablas en la base de datos**.

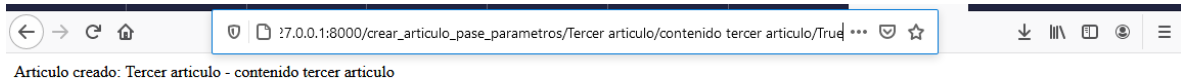
Hacer cambios en la estructura de las tablas y de modelos:

- Despues de realizar los cambios en los campos de los modelos en **models.py** se tiene que realizar lo siguiente:
- En el símbolo del sistema ubicarse en la carpeta del proyecto.
- Teclear **python manage.py makemigrations**, después.
- Teclear **python manage.py sqlmigrate NOMBRE_DE_LA_APP** y los primeros 4 **numeros de la migration** por ejemplo: **0002**, después.
- Teclear **python manage.py migrate** es ahí donde se aplica los cambios de las **tablas en la base de datos**.

Guardar registros usando modelos:

```
def crear_articulo(request):  
    articulo = Article(  
        title = 'Primer articulo',  
        content = 'Contenido del articulo',  
        public = True  
    )  
  
    articulo.save() ←--- ESTA ES LA LINEA PARA GUARDAR LOS REGISTROS  
  
    return HttpResponse("Usuario creado: ")
```

Guardar registros usando parámetros de la URL :



Views.py

```
def crear_articulo_pase_parametros(request, title, content, public):
    articulo = Article(
        title = title,
        content = content,
        public = public
    )

    articulo.save()

    return HttpResponse(f"Articulo creado: {articulo.title} - {articulo.content}")
```

urls.py

```
path('crear_articulo_pase_parametros/<str:title>/<str:content>/<str:public>', views.crear_articulo_pase_parametros, name="crear_articulo_pase_parametros"), # GUARDAR ARTICULO CON PASE DE PARAMETROS
```

Sacar datos y elementos de la base de datos

Views.py

```
def verarticulo(request):
    try:
        articulo = Article.objects.get(id=1,public=True) # Buscar por medio del indice en lugar el ID puede ir el nombre del campo la (,) se usa como AND en un WHERE DE LA CONSULTA
        #articulo = Article.objects.get(title="Primer articulo") # Buscar por medio del indice en lugar el ID puede ir el nombre del campo
        response = f"Articulo: {articulo.title}"
    except:
        response = f"Articulo: No encontrado"

    return HttpResponse(response)
```

urls.py

```
path('verarticulo/', views.verarticulo, name="verarticulo"),
```


Actualizar registros en la base de datos

Views.py

```
def editar_articulo(request, id):
    articulo = Article.objects.get(pk=id)
    articulo.title="Segundo Modificado"
    articulo.content="Contenido Segundo Modificado"
    articulo.public= False

    articulo.save()

    return HttpResponse(f"Articulo editado: {articulo.title} -
{articulo.content}")
```

urls.py

```
path('editar_articulo/<int:id>', views.editar_articulo, name="editar_
articulo"),
```

Listar todos los artículos

Views.py

```
def listar_articulos(request):
    articulos = Article.objects.all()

    return render(request, 'listar_articulos.html', {
        'articulos': articulos
    })
```

Urls.py

```
path('listar_articulos/', views.listar_articulos, name="listar_articulos"),
```

listar_articulos.html

```
{% extends 'base.html' %}
{% block title %} Listado de articulos {% endblock %}
{% block content %}

<h1>Listado de articulos</h1>
    <ul>
        {% for articulo in articulos %}
            <li>{{ articulo.id }}. {{ articulo.title }}</li>
        {% endfor %}
    </ul>
{% endblock %}
```

Limit y Order By

Views.py

```
def listar_articulos(request):
    '''articulos = Article.objects.all()'''
    articulos = Article.objects.order_by('title')
    '''articulos = Article.objects.order_by('-id')''' # el (-
id): ordena por id de manera descendente
    '''articulos = Article.objects.order_by('-
id')[:3]''' # el [:3] : solo muestra los 3 elementos es un limit
    '''articulos = Article.objects.order_by('-
id')[3:7]''' # el [3:7] : solo muestra los elementos del 3 al 7 es u
n limit

    return render(request, 'listar_articulos.html', {
        'articulos': articulos
    })
```

Eliminar registros :

Views.py

```
def borrar_articulo(request, id):
    articulo = Article.objects.get(id=id)
    articulo.delete() ←----- ESTO ELIMINAR LOS REGISTROS

    return redirect('listar_articulos')←--- ESTO REDIRECCIONA
```

urls.py

```
path('borrar_articulo/<int:id>', views.borrar_articulo, name="borrar_
articulo"),
```

Consultas con condiciones filter y lookups :

views.py

```
def listar_articulos(request):
    articulos = Article.objects.all()
    '''articulos = Article.objects.order_by('title')'''
    '''articulos = Article.objects.order_by('-id')''' # el (-
id): ordena por id de manera descendente
    '''articulos = Article.objects.order_by('-
id')[:3]''' # el [:3] : solo muestra los 3 elementos es un limit
    '''articulos = Article.objects.order_by('-
id')[3:7]''' # el [3:7] : solo muestra los elementos del 3 al 7 es u
n limit

    #articulos = Article.objects.filter(title="Primer articulo batman
") # Filter la busqueda lo hace que contenga toda la cadena
    #articulos = Article.objects.filter(title__contains="batman") # d
espues de filter se pone en parentesis el campo continuando con __ (d
oble guion bajo) a eso se le llama LOOKUPS la busqueda lo hace que co
ntenga la cadena BATMAN no es necesario toda la cadena es como un LIK
E en MYSQL
    #articulos = Article.objects.filter(title__exact="batman") # desp
ues de filter se pone en parentesis el campo continuando con __ (dobl
e guion bajo) a eso se le llama LOOKUPS la busqueda lo hace que conte
nga la cadena EXACTA (exact) BATMAN
    articulos = Article.objects.filter(id__lte=3, title__contains="ba
tman") # busca los id = 3 o mayor que 3 y que contengan batman

    return render(request, 'listar_articulos.html', {
        'articulos': articulos
    })
```

Exclude

Views.py

```
def listar_articulos(request):
    articulos = Article.objects.filter(title__exact="batman").exclude(
        public=True) # el EXCLUDE sirve para excluir los registros que tenga
    n ese valor

    return render(request, 'listar_articulos.html', {
        'articulos': articulos
    })
```

Ejecutar SQL desde Django

Views.py

```
def listar_articulos(request):
    articulos = Article.objects.raw("SELECT * FROM app_article where
title='Articulo 1' and public=0")

    return render(request, 'listar_articulos.html', {
        'articulos': articulos
    })
```

OR en consultas con el ORM

Views.py

```
from django.shortcuts import render, HttpResponseRedirect, redirect
from app.models import Article, Category
from django.db.models import Q ←----- Q se tiene que importar para
que funcione el OR
```

```
def listar_articulos(request):

    articulos = Article.objects.filter(Q(title__contains="2") | Q(title__contains="3")) # El Q es el OR en consultas SQL

    return render(request, 'listar_articulos.html', {
        'articulos': articulos
    })
```

Recibir datos del formulario por GET

Create_article.html

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

{% block title %} Formularios en Django {% endblock %}

{% block content %}
<h1>Formularios en Django</h1>
<form action="{% url 'save_article' %}" method="GET"> <---- AQUÍ SE
USA EL METODO DE ENVIO GET Y EN ACTION MANDO LA RUTA HACIA DONDE VAN
A IR LOS DATOS EN ESTE CASO A SAVE_ARTICLE
    <label for="title">Titulo</label>
    <input type="text" name="title" placeholder="Introduce el titulo
del articulo">

    <label for="content">Contenido</label>
    <textarea name="content"></textarea>

    <label for="public">Publicado</label>
    <select name="public">
        <option value="1">SI</option>
        <option value="0">NO</option>
    </select>
    <input type="submit" value="Guardar">

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

Views.py

```
def save_article(request):

    if request.method == 'GET': <--- aquí recibo por GET

        title = request.GET['title'] <--- aquí recibo por GET
        content = request.GET['content'] <--- aquí recibo por GET
        public = request.GET['public'] <--- aquí recibo por GET

        articulo = Article(
            title = title,
            content = content,
            public = public
        )

        articulo.save()

        return HttpResponse(f"Articulo creado: {articulo.title} - {articulo.content}")

    else:

        return HttpResponse(f"No se pudo crear el articulo")
```


Recibir datos del formulario por POST

Create_article.html

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

{% block title %} Formularios en Django {% endblock %}

{% block content %}
<h1>Formularios en Django</h1>
<form action="{% url 'save_article' %}" method="POST"> ← enviar por
método POST

    {% csrf_token %} ←---- ESTO TIENE QUE IR A LA FUERZA SI SE USA EL
    ENVIO DE VARIABLES POR EL MÉTODO POST

    <label for="title">Titulo</label>
    <input type="text" name="title" placeholder="Introduce el titulo
del articulo">

    <label for="content">Contenido</label>
    <textarea name="content"></textarea>

    <label for="public">Publicado</label>
    <select name="public">
        <option value="1">SI</option>
        <option value="0">NO</option>
    </select>
    <input type="submit" value="Guardar">

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

Views.py

```
def save_article(request):  
  
    if request.method == 'POST': ← RECIBIR POR POST  
  
        title = request.POST['title'] ← RECIBIR POR POST  
        content = request.POST['content'] ← RECIBIR POR POST  
        public = request.POST['public'] ← RECIBIR POR POST  
  
        articulo = Article(  
            title = title,  
            content = content,  
            public = public  
        )  
  
        articulo.save()  
  
        return HttpResponseRedirect(f"Articulo creado: {articulo.title} -  
{articulo.content}")  
  
    else:  
  
        return HttpResponseRedirect(f"No se pudo crear el articulo")
```

Formularios basados en clases

Se crea el archivo forms.py a nivel de admin.py

Forms.py

```
from django import forms

class FormArticle(forms.Form):
    title = forms.CharField(
        label = "Titulo"
    )

    contenido = forms.CharField(
        label = "Contenido",
        widget=forms.Textarea
    )

    public_options = [
        (1, 'Si'),
        (0, 'No')
    ]

    public = forms.TypedChoiceField(
        label= "Publicado",
        choices = public_options
    )
```

Views.py

```
from django.shortcuts import render, HttpResponseRedirect, redirect
from app.models import Article, Category
from django.db.models import Q
from app.forms import FormArticle ←- ESTO SE IMPORTA PARA USO DE FORMULARIOS

def create_full_article(request):
    formulario = FormArticle()
    return render(request, 'create_full_article.html', {
        'form': formulario
    })
```

Create_full_article.html

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

{% block title %} Formularios en Django {% endblock %}

{% block content %}
<h1>Formularios en Django</h1>
<form action="" method="POST">

    {% csrf_token %}

    {{ form.as_ul }} ← LA DIFERENCIA ENTRE UN HTML NORMAL Y UN FORM ES ESTO
<!--
    form.as_ul sirve para poner cada elemento en forma de lista y así ha
    y mas opciones -->

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

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

Urls.py

```
path('create_full_article/', views.create_full_article, name="create_full_article"),
```

Recibir datos y guardar el formulario (Django Form API) :

Views.py

```
def create_full_article(request):

    if request.method == 'POST':
        formulario = FormArticle(request.POST)
        if formulario.is_valid(): <- VALIDA EL FORMULARIO
            data_form = formulario.cleaned_data <- LIMPIA LOS DATOS
            DEL FORMULARIO QUE NO PASE BASURA
            title = data_form.get('title') <- ASI SE OBTIENEN
            content = data_form.get('content') <- ASI SE OBTIENEN
            public = data_form.get('public') <- ASI SE OBTIENEN

            articulo = Article(
                title = title,
                content = content,
                public = public
            )

            articulo.save()
            return redirect('listar_articulos')

    else:
        formulario = FormArticle()

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

Validar formularios en django (BUSCAR VALIDATOR EN LA DOCUMENTACION DE DJANGO) :

Views.py

```
from django import forms
from django.core import validators # LIBRERIA PARA VALIDAR FORMULARIOS
```

```
class FormArticle(forms.Form):
    title = forms.CharField(
        label = "Titulo",
        max_length=40,
        required=True,
        widget=forms.TextInput(
            attrs={
                'placeholder':'Mete el titulo',
                'class':'title_form_article'
            }
        ),
        validators=[
            validators.MinLengthValidator(4,'El titulo es demasiado corto')← VALIDA QUE EL TITULO TENGA MAS DE 4 DE LONGITUD
            #validators.RegexValidator('^[A-Za-z0-9]*$', 'El titulo esta mal formado','invalid_title')
        ]
    )
```

Create_full_article.html

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

{% block title %} Formularios en Django {% endblock %}

{% block content %}
<h1>Formularios en Django</h1>

{% if form.errors %} <-- AQUÍ MANDA EL MENSAJE DE ERROR DE VALIDACION
    <strong>
        Hay errores en el formulario
    </strong>
{% endif %}

<form action="" method="POST">

    {% csrf_token %}

    {{ form.as_ul }} <!--
    form.as_ul sirve para poner cada elemento en forma de lista y así ha
    y mas opciones -->

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

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

Mensajes flash / sesiones flash (BUSCAR MESSAGES EN LA DOCUMENTACION DE DJANGO) :

Views.py

```
from django.shortcuts import render, HttpResponseRedirect, redirect
from app.models import Article, Category
from django.db.models import Q
from app.forms import FormArticle
from django.contrib import messages # SE IMPORTA ESTA LIBRERIA PARA M
ANDAR LOS MENSAJES
```



```

def create_full_article(request):

    if request.method == 'POST':
        formulario = FormArticle(request.POST)
        if formulario.is_valid():
            data_form = formulario.cleaned_data
            title = data_form.get('title')
            content = data_form.get('content')
            public = data_form.get('public')

            articulo = Article(
                title = title,
                content = content,
                public = public
            )

            articulo.save()

            # crear mensaje flash (sesion que solo se muestra 1 vez,
            es el aviso que se guardo el registro)
            messages.success(request, f'Has creado correctamente el ar
            ticulo {articulo.id}') ← ESTO SON LOS MENSAJES QUE SE MUESTRAN PARA
            AVISAR QUE YA SE GUARDO EL REGISTRO

            return redirect('listar_articulos')

        else:
            formulario = FormArticle()

    return render(request, 'create_full_article.html', {
        'form': formulario
    })

```

Listar_articulos.html

```
{% extends 'base.html' %}
{% block title %} Listado de articulos {% endblock %}
{% block content %}

<h1>Listado de articulos</h1>

{% if messages %} ←- ESTO ES PARA MANDAR LOS MENSAJES
    {% for message in messages %}
        <div style="background-color: red;">
            {{message}}
        </div>
    {% endfor %}
{% endif %}

<ul>
    {% for articulo in articulos %}
        <li>{{articulo.id}}. {{articulo.title}}</li>
        <p>{{articulo.content}}
        <a href="{% url 'borrar_articulo' articulo.id %}">Elimina
r</a>

        </p>
        <br>
    {% endfor %}
</ul>
{% endblock %}
```

Primeros pasos con el panel de administración de django:

- En el símbolo del sistema tecleamos: **python manage.py createsuperuser (enter), USUARIO: monchis, CLAVE: monchis1981**
- **Poner nombre de usuario, correo y contraseña**
- **Esto es para crear el superusuario para entrar al panel de adminstracion.**
- **Para entrar al panel de administración es: 127.0.0.1:8000/admin**
-

Admin.py

```
from django.contrib import admin
from .models import Article,Category ← Se importan los modelos

# Register your models here.

admin.site.register(Article) ← Se registra cada modelo
admin.site.register(Category) ← Se registra cada modelo
```

Settings.py

Al final de settings.py hay que colocar este código para que carguen los css del panel de administración.

```
#STATIC_URL = '/static/'

#BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

STATIC_ROOT = os.path.join(BASE_DIR, 'static')
STATIC_URL = '/static/'
```

Clase META en los modelos de Django 3 (buscar en google meta clase django)

Models.py

```
from django.db import models

# Create your models here.
class Article(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    image = models.ImageField(default='null')
    public = models.BooleanField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta: <- Uso del class Meta
        verbose_name = "Articulo" <- Titulo en singular
        verbose_name_plural = "Articulos" <- Titulo en plural
        ordering = ['created_at'] <- Ordenar por fecha

class Category(models.Model):
    name = models.CharField(max_length=100)
    description = models.CharField(max_length=250)
    created_at = models.DateField()

    class Meta:
        verbose_name = "Categoria"
        verbose_name_plural = "Categorias"
```

Cambiar nombre de las apps

Apps.py

```
from django.apps import AppConfig

class AppConfig(AppConfig):
    name = 'app'
    verbose_name = 'Mi primera aplicacion' ←- AQUÍ SE PONE EL NUEVO
NOMBRE
```

settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app.apps.AppConfig' # AQUÍ SE CAMBIA EL NOMBRE DE LA APP
]
```

Metodo mágico para imprimir objetos (modelos), buscar en google (magic methods python)

Models.py

```
# Create your models here.
class Article(models.Model):
    title = models.CharField(max_length=100,verbose_name="Titulo")
    content = models.TextField(verbose_name="Contenido")
    image = models.ImageField(default='null',verbose_name="Imagen")
    public = models.BooleanField(verbose_name="Publicado")
    created_at = models.DateTimeField(auto_now_add=True,verbose_name="Titulo")
    updated_at = models.DateTimeField(auto_now=True,verbose_name="Titulo")

    class Meta:
        verbose_name = "Articulo"
        verbose_name_plural = "Articulos"
        ordering = ['created_at']

    def __str__(self): ← METODO MAGICO
        if self.public:
            publico = "(Publicado)"
        else:
            publico = "(Privado)"
        return f"{self.title} ({publico})"
```

Mostrar campos de solo lectura

Admin.py

```
from django.contrib import admin
from .models import Article, Category

# Register your models here.

class ArticleAdmin(admin.ModelAdmin): ←- SE CREA LA CLASE
    readonly_fields = ('created_at', 'updated_at') ←- AQUÍ SE INDICAN
    QUE CAMPOS SERAN DE SOLO LECTURA

admin.site.register(Article, ArticleAdmin) ←- se agrega en el
admin.site.register la clase ArticleAdmin
admin.site.register(Category)
```

Subir imágenes en Django

Settings.py al final

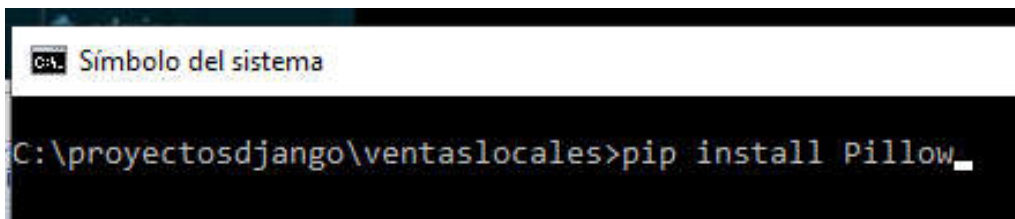
```
# MEDIA FILES ←aquí se configura la ruta donde se guardaran las
imágenes y se tiene que crear en raíz del proyecto la carpeta MEDIA

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


Models.py

```
# Create your models here.
class Article(models.Model):
    title = models.CharField(max_length=100,verbose_name="Titulo")
    content = models.TextField(verbose_name="Contenido")
    image = models.ImageField(default='null',verbose_name="Imagen", u
pload_to="articles") # < se crea la carpeta ARTICLES dentro de MEDIA
    public = models.BooleanField(verbose_name="Publicado")
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

En el símbolo del sistema detenemos el servidor y tenemos que instalar el paquete Pillow



```
C:\proyectosdjango\ventaslocales>pip install Pillow_
```

urls.py

```
from django.contrib import admin
from django.urls import path
from django.conf import settings # <IMPORTO SETTINGS.PY AQUÍ para
mandar a llamar las variables al final de urls.py en la configuracion
```

```
# Configuracion para cargar imágenes y verlos en el navegador en una
ventana aparte como una url
if settings.DEBUG:
    from django.conf.urls.static import static
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.
MEDIA_ROOT)
```

Mostrar imágenes subidas en la web

Listar_articulos.html

```
{% extends 'base.html' %}
{% block title %} Listado de articulos {% endblock %}
{% block content %}

<h1>Listado de articulos</h1>

{% if messages %}
    {% for message in messages %}
        <div style="background-color: red;">
            {{message}}
        </div>
    {% endfor %}
{% endif %}

<ul>
    {% for articulo in articulos %}
        {% if articulo.image.url|length > 0 and articulo.image !=
'null' %}<- ESTO ES PARA MOSTRAR LAS IMAGENES
        <- ESTO ES PARA
MOSTRAR LAS IMAGENES
        {% endif %}<- ESTO ES PARA MOSTRAR LAS IMAGENES

        <li>{{articulo.id}}. {{articulo.title}}</li>
        <p>{{articulo.content}}
        <a href="{% url 'borrar_articulo' articulo.id %}">Elimina
r</a>

        </p>
        <br>
    {% endfor %}
</ul>
{% endblock %}
```

Cambiar el titulo del panel

admin.py

Al final del archivo se pone esto

```
# Configurar el titulo del panel
admin.site.site_header = "Panel de Administracion - Ventas Locales"
```

Cambiar el subtítulo del panel (pestaña del navegador)

admin.py

Al final del archivo se pone esto

```
# Configurar el sbutitulo del panel

admin.site.site_title = "Panel de Administracion - Ventas Locales"
admin.site.index_title = "Panel de Gestion"
```

Menu dinamico y context processors.

Un context processors es una opción que nos permite tener información en las templates.

Se crea en la carpeta raíz de la app (ejemplo pages) el archivo context_processors.py

Context_processor.py

```
from pages.models import Page

def get_pages(request):
    pages = Page.objects.values_list('id','title','slug')
    # SOLO ME DEVUELVE LOS 3 CAMPOS EN LUGAR DE USAR ALL() ME MOSTRARIA TODOS LOS CAMPOS

    return {
        'pages': pages
    }
```

settings.py

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [],  
        'APP_DIRS': True,  
        'OPTIONS': {  
            'context_processors': [  
                'django.template.context_processors.debug',  
                'django.template.context_processors.request',  
                'django.contrib.auth.context_processors.auth',  
                'django.contrib.messages.context_processors.messages'  
            ],  
            'pages.context_processors.get_pages', ← AQUÍ SE AGREGA  
            EL CONTEXT PROCESSOR QUE SE CREO  
        },  
    },  
]
```

Mejorar la configuración de rutas.

INVESTIGARLO EN GOOGLE COMO FUNCION include en urls.py

Urls.py (de la raíz del proyecto principal)

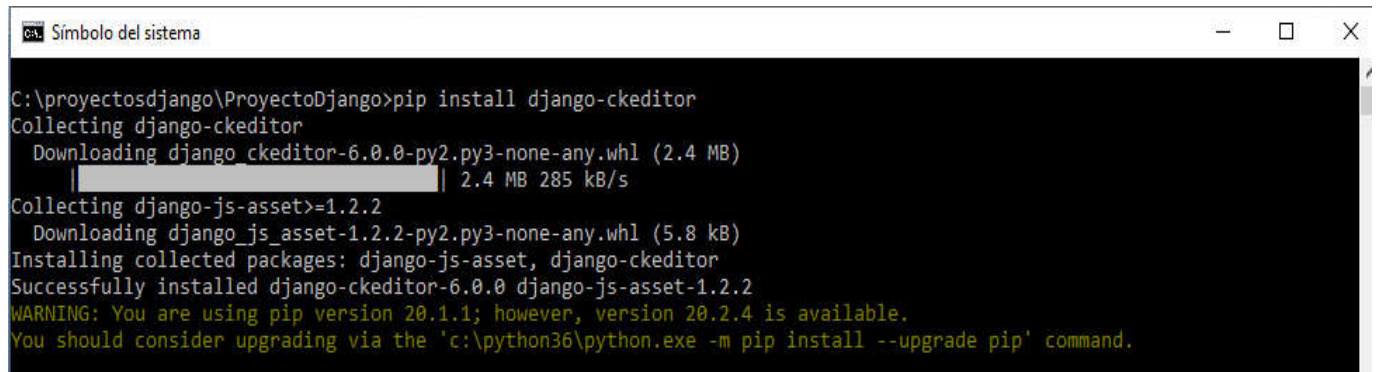
```
from django.contrib import admin
from django.urls import path
from mainapp import views
from pages import views as page_views # ESTO ES PARA IMPORTAR LAS VIEWS DE LA APP PAGES Y NO TENER PROBLEMAS
#CON LA IMPORTACION DE LA LINEA ANTERIOR
from django.conf.urls import include # Esto se usa para incluir las rutas de otros archivos urls.py de las aplicaciones
from django.conf import settings

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index, name="index"),
    path('inicio/', views.index, name="inicio"),
    path('sobre_nosotros/', views.about, name="about"),
    path('pagina/<str:slug>', page_views.page, name="page"),
    #path('', include('pages.urls')),
    path('', include('blog.urls')), # Aquí se incluyen las rutas de la aplicación blog que se encuentra en la carpeta blog
    path('register_page/', views.register_page, name="register_page"),
    ,
    path('login_page/', views.login_page, name="login_page"),
    path('logout_user/', views.logout_user, name="logout_user"),
]

# Ruta imagenes
# Configuración para cargar imagenes
if settings.DEBUG:
    from django.conf.urls.static import static
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Usar un editor de texto enriquecido en Django.

Se tiene que instalar el paquete **django-ckeditor**



```
Simbolo del sistema
C:\proyectosdjango\ProyectoDjango>pip install django-ckeditor
Collecting django-ckeditor
  Downloading django_ckeditor-6.0.0-py2.py3-none-any.whl (2.4 MB)
    | 2.4 MB 285 kB/s
Collecting django-js-asset>=1.2.2
  Downloading django_js_asset-1.2.2-py2.py3-none-any.whl (5.8 kB)
Installing collected packages: django-js-asset, django-ckeditor
Successfully installed django-ckeditor-6.0.0 django-js-asset-1.2.2
WARNING: You are using pip version 20.1.1; however, version 20.2.4 is available.
You should consider upgrading via the 'c:\python36\python.exe -m pip install --upgrade pip' command.
```

Settings.py

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'ckeditor', ← Tenemos que agregar la aplicación CKEDITOR
    'mainapp',
    'pages.apps.PagesConfig'
]
```

models.py

```
from django.db import models
from ckeditor.fields import RichTextField ← Importamos CKEDITOR
```

models.py

```
# Create your models here.
class Page(models.Model):
    title = models.CharField(max_length=50, verbose_name="Titulo de l
a pagina")
    content = RichTextField(verbose_name="Contenido de la pagina")←
Aquí ya no lleva models solo ira RichTextField

    slug = models.CharField(unique=True, max_length=150, verbose_name
="Url amigable") # UNIQUE sirve para valor UNICO evita repetir VALORE
S
    visible = models.BooleanField(verbose_name="¿Visible?")
    created_at = models.DateTimeField(auto_now_add=True, verbose_name
="Creado el")
    updated_at = models.DateTimeField(auto_now=True, verbose_name="Ac
tualizado el")

    class Meta:
        verbose_name = "Pagina"
        verbose_name_plural = "Paginas"

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


En el template (page.html)

```
{% extends 'layouts/layout.html' %}

{% block title %} {{page.title}} {% endblock %}

{% block content %}
    <h1 class="title">{{page.title}}</h1>
    <span class="date">{{page.created_at}}</span>
    <p>
        {{page.content|safe}} <---el | y safe se usan para que
        muestre caracteres especiales html e imágenes del editor ckeditor
    </p>
{% endblock %}
```

Modificar el modelo y agregar orden de paginas.

Models.py

```
# Create your models here.
class Page(models.Model):
    title = models.CharField(max_length=50, verbose_name="Titulo de la pagina")
    content = RichTextField(verbose_name="Contenido de la pagina")
    slug = models.CharField(unique=True, max_length=150, verbose_name="Url amigable") # UNIQUE sirve para valor UNICO evita repetir VALORES
    order = models.IntegerField(default=0, verbose_name="Ordenacion")
    ← agregamos el campo order
    visible = models.BooleanField(verbose_name="¿Visible?")
    created_at = models.DateTimeField(auto_now_add=True, verbose_name="Creado el")
    updated_at = models.DateTimeField(auto_now=True, verbose_name="Actualizado el")

    class Meta:
        verbose_name = "Pagina"
        verbose_name_plural = "Paginas"

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

En consola ejecutamos:

- Python manage.py makemigrations
- Despues ejecutamos en consola: python manage.py sqlmigrate NOMBREAPP(pages) y los 3 primeros numero de la migración 0002
- Despues python manage.py migrate

Context_processors.py

```
from pages.models import Page

def get_pages(request):
    #pages = Page.objects.values_list('id','title','slug') # SOLO ME
    #DEVUELVE LOS 3 CAMPOS EN LUGAR DE USAR ALL() ME MOSTRARIA TODOS LOS C
    #AMPOS
    #pages = Page.objects.filter(visible=True).values_list('id','titl
    #e','slug') # ESTE SOLO MUESTRA LAS PAGINAS QUE SON VISIBLE
    pages = Page.objects.filter(visible=True).order_by('order').value
    s_list('id','title','slug') # FILTRA SOLO LAS PAGINAS VISIBLES Y LOS
    #ORDENA DE ACUERDO AL CAMPO ORDER
    return {
        'pages': pages
    }
```

Relaciones entre los modelos.

models.py

```
class Article(models.Model):
    title = models.CharField(max_length=150, verbose_name='Titulo')
    content = RichTextField(verbose_name='Contenido')
    image = models.ImageField(default='null', verbose_name='Imagen')
    public = models.BooleanField(verbose_name='Publicado')
    user = models.ForeignKey(User, verbose_name='Usuario', on_delete=
models.CASCADE)# Guarda el ID DEL USUARIO que ha creado el articulo e
sta es la RELACION ENTRE MODELOS (TIENE EL OBJETO COMPLETO DEL MODELO
USUARIO), on_delete=CASCADE borra todos los elementos en cascada es
decir al momento de borrar el usuario borrara todos los articulos que
haya guardado
    categories = models.ManyToManyField(Category, verbose_name='Categ
orias', blank=True)# ManyToManyField relacion MUCHO A MUCHOS
    created_at = models.DateTimeField(auto_now_add=True, verbose_name
='Creado el')
    update_at = models.DateTimeField(auto_now=True, verbose_name='Edi
tado el')
```

Guardar usuario con el inicio de sesion.

Models.py

```
class Article(models.Model):
    title = models.CharField(max_length=150, verbose_name='Titulo')
    content = RichTextField(verbose_name='Contenido')
    image = models.ImageField(default='null', verbose_name='Imagen')
    public = models.BooleanField(verbose_name='Publicado')
    user = models.ForeignKey(User, editable=False, verbose_name='Usua
rio', on_delete=models.CASCADE)# Ponemos editable=False para que no
se muestre el combo de los usuarios
    categories = models.ManyToManyField(Category, verbose_name='Categ
orias', blank=True)# ManyToManyField relacion MUCHO A MUCHOS
    created_at = models.DateTimeField(auto_now_add=True, verbose_name
='Creado el')
    update_at = models.DateTimeField(auto_now=True, verbose_name='Edi
tado el')
```

admin.py

```
class ArticleAdmin(admin.ModelAdmin):
    readonly_fields = ('user', 'created_at', 'update_at')

    def save_model(self, request, obj, form, change): ←Tenemos que mandar
todos estos parametros
        if not obj.user_id: ←Si no mandamos ningún ID_USUARIO
            obj.user_id = request.user.id←Entonces nosotros los
obtenemos de acuerdo al user_id de la sesión con el método request
            obj.save()
```

Subconsultas en django.

processor.py (dentro de la carpeta blog)

```
from blog.models import Category,Article

def get_category(request):
    categories_in_use = Article.objects.filter(public=True).values_list('categories',flat=True) # se obtiene los ids de las categorias que tienen al menos un articulo
    categories = Category.objects.filter(id__in=categories_in_use).values_list('id','name') # FILTRA SOLO LAS CATEGORIAS QUE TIENEN ARTICULOS <-- ESTO ES LAS SUBCONSULTAS EN DJANGO
    con filter(id__in=categories_in_use)

    return {
        'categories': categories,
        'ids': categories_in_use
    }
```

Mostrar error 404.

views.py (dentro de la carpeta blog)

```
from django.shortcuts import render, get_object_or_404 # get_object_or_404 <-- este es para mandar mensajes de error 404
```

```
from blog.models import Article, Category
```

```
def category(request, category_id):  
    #category = Category.objects.get(id=category_id)  
    category = get_object_or_404(Category, id=category_id) # Manda el aviso del error 404  
    return render(request, 'categories/category.html', {  
        'category': category  
    })
```

Datos relacionados entre modelos y relaciones inversas

models.py (dentro de la carpeta blog)

```
class Article(models.Model):
    title = models.CharField(max_length=150, verbose_name='Titulo')
    content = RichTextField(verbose_name='Contenido')
    image = models.ImageField(default='null', verbose_name='Imagen',
upload_to="articles")
    public = models.BooleanField(verbose_name='Publicado')
    user = models.ForeignKey(User, editable=False, verbose_name='Usua
rio', on_delete=models.CASCADE)
    categories = models.ManyToManyField(Category, verbose_name='Categ
orias', blank=True, related_name="articles")# ManyToManyField relacio
n MUCHO A MUCHOS (related_name="articles" <--
esto indica la relacion con los articulos
    created_at = models.DateTimeField(auto_now_add=True, verbose_name
='Creado el')
    update_at = models.DateTimeField(auto_now=True, verbose_name='Edi
tado el')

    class Meta:
        verbose_name = 'Articulo'
        verbose_name_plural = 'Articulos'

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


category.html (dentro de la carpeta blog)

```
{% extends 'layouts/layout.html' %}

{% block title %} {{category.name}} {% endblock %}

{% block content %}
    <h1 class="title">{{category.name}}</h1>
    <p>{{category.description}}</p>
    {% include 'articles/articles.html' with articles=articles %}<!--
    with articles = articles ES lo mismo que si le estuviéramos pasando
    la variable ARTICLES como lo envía en views.py -->
    {% include 'articles/articles.html' with articles=category.articles.all %}<!-- RELACION INVERSA -->

{% endblock %}
```

Personalizar listados.

admin.py (dentro de la carpeta pages)

```
from django.contrib import admin
from .models import Page

# Configuración extra
class PageAdmin(admin.ModelAdmin):
    readonly_fields = ('created_at', 'updated_at')
    search_fields = ('titulo', 'content') # con esto se activa el buscador

# Register your models here.
admin.site.register(Page, PageAdmin)
```

Escoja Pagina a modificar

AÑADIR PAGINA +

Accion: seleccionados 0 de 3

- ☐ PAGINA
- ☐ Trabaja con nosotros
- ☐ Servicios
- ☐ Sobre Nosotros

3 Paginas

```

from django.contrib import admin
from .models import Page

# Configuración extra
class PageAdmin(admin.ModelAdmin):
    readonly_fields = ('created_at', 'updated_at')
    search_fields = ('titulo', 'content') # con esto se activa el buscador
    list_filter = ('visible',) # filtrame con los campos

# Register your models here.
admin.site.register(Page, PageAdmin)

```

Escoja Pagina a modificar

Q

Accion: seleccionados 0 de 3

- ☐ PAGINA
- ☐ Trabaja con nosotros
- ☐ Servicios
- ☐ Sobre Nosotros

3 Paginas

FILTRO

Por ¿Visible?

☐ Todo
☐ Si
☐ No

```

from django.contrib import admin
from .models import Page

# Configuración extra
class PageAdmin(admin.ModelAdmin):
    readonly_fields = ('created_at', 'updated_at')
    search_fields = ('titulo', 'content') # con esto se activa el buscador
    list_filter = ('visible',) # filtrame con los campos
    list_display = ('title', 'visible', 'created_at') # me muestra los campos que quiero ver en esa lista en las cabeceras de columna
    ordering = ('created_at') # Ordenar por el campo created_at

# Register your models here.
admin.site.register(Page, PageAdmin)

```

Escoja Pagina a modificar

q

Accion: seleccionados 0 de 3

<input type="checkbox"/>	TITULO DE LA PAGINA	¿VISIBLE?	CREADO EL
<input type="checkbox"/>	Trabaja con nosotros	<input checked="" type="checkbox"/>	25 de Octubre de 2020 a las 14:01
<input type="checkbox"/>	Servicios	<input checked="" type="checkbox"/>	25 de Octubre de 2020 a las 14:01
<input type="checkbox"/>	Sobre Nosotros	<input checked="" type="checkbox"/>	25 de Octubre de 2020 a las 14:00

3 Paginas

FILTRO

Por ¿Visible?

Todo

Si

No

Como hacer una paginación en Django 3.

views.py (dentro de la carpeta blog)

```
from django.shortcuts import render, get_object_or_404 # get_object_or_404 <-- este es para mandar mensajes de error 404
from django.core.paginator import Paginator # Importamos para realizar la paginacion
from blog.models import Article, Category

# Create your views here.
def articles(request):
    # Sacar articulos
    articles = Article.objects.all()

    #Paginar los articulos
    paginator = Paginator(articles, 2) # El numero 2 es cuantos articulos mostrara por pagina

    #Recoger numero pagina
    page = request.GET.get('page')
    page_articles = paginator.get_page(page)

    return render(request, 'articles/list.html', {
        'title': 'Articulos',
        'articles': page_articles
    })
```

Pagination.html (dentro de la carpeta blog)

```
<div>
  <h1>Paginacion</h1>

  {% if articles.has_previous %}
    <a href="?page=1">&laquo; Primero</a>
    <a href="?page={{ articles.previous_page_number }}">Anterior<
/a>
  {% endif %}

  <span>Pagina {{ articles.number }} de {{ articles.paginator.num_p
ages }}</span>

  {% if articles.has_next %}
    <a href="?page={{ articles.next_page_number }}">Siguiente</a>
    <a href="?page={{ articles.paginator.num_pages }}">Ultima &ra
quo;</a>
  {% endif %}

</div>
```

Formulario de registro de usuarios.

forms.py (dentro de la carpeta mainapp)

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

from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User

class RegisterForm(UserCreationForm):
    class Meta:
        model = User
        fields = ['username', 'email', 'first_name', 'last_name', 'password1', 'password2'] # Aqui mostramos los campos que se mostraran en el formulario
```

views.py (dentro de la carpeta mainapp)

```
from django.shortcuts import render, redirect
from django.contrib.auth.forms import UserCreationForm # <-
Se importa esto para el formulario de registro de usuarios
from mainapp.forms import RegisterForm # Se importa de forms.py
```

```
def register_page(request):

    register_form = RegisterForm() # Creacion del objeto formulario d
e registro

    if request.method == 'POST': # Preguntamos si viene por POST
        register_form = RegisterForm(request.POST) # Se le asigna a l
a variable register_form el objeto con los valores de la request

        if register_form.is_valid(): # Preguntamos si es valido el fo
rmulario
            register_form.save()      # Si es valido se guarda
            return redirect('inicio') # Luego de guardar se redirige
a la pagina de inicio

        return render(request, 'users/register.html', {
            'title': 'Registro',
            'register_form': register_form # Enviamos hacia el template l
a variable register_form
        })
```


Mensaje flash en el registro de usuario

views.py (dentro de la carpeta mainapp)

```
from django.shortcuts import render, redirect
from django.contrib import messages # mensaje flash en el registro
from django.contrib.auth.forms import UserCreationForm # Se importa esto para el formulario de registro de usuarios
from mainapp.forms import RegisterForm # Se importa de forms.py
```

```
def register_page(request):

    register_form = RegisterForm() # Creacion del objeto formulario de registro

    if request.method == 'POST': # Preguntamos si viene por POST
        register_form = RegisterForm(request.POST) # Se le asigna a la variable register_form el objeto con los valores de la request

        if register_form.is_valid(): # Preguntamos si es valido el formulario
            register_form.save() # Si es valido se guarda
            messages.success(request, 'Te has registrado correctamente') # Mensaje que guardo correctamente
            return redirect('inicio') # Luego de guardar se redirige a la pagina de inicio

        return render(request, 'users/register.html', {
            'title': 'Registro',
            'register_form': register_form # Enviamos hacia el template la variable register_form
        })
```

Login de usuarios

views.py (dentro de la carpeta mainapp)

```
from django.shortcuts import render, redirect
from django.contrib import messages # mensaje flash en el registro
from django.contrib.auth.forms import UserCreationForm # Se importa esto para el formulario de registro de usuarios
from mainapp.forms import RegisterForm # Se importa de forms.py
from django.contrib.auth import authenticate, login, logout # Importamos estos modulos para el FORMULARIO DE LOGIN
```

```
def login_page(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('inicio')
        else:
            messages.warning(request, 'No te has identificado correctamente')

    return render(request, 'users/login.html', {
        'title': 'Identificate'
    })
```

Urls.py

```
path('login_page/', views.login_page, name="login_page"),
```

login.html

```
{% extends 'layouts/layout.html' %}

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

{% block content %}
    <h1>{{title}}</h1>

    {% if messages %}<!--MENSAJE DE ALERTA QUE SE GUARDO-->
    {% for message in messages %}<!--MENSAJE DE ALERTA QUE SE GUARDO-->
->
        <div class="alert alert-success"><!--
MENSAJE DE ALERTA QUE SE GUARDO-->
            {{message}}<!--MENSAJE DE ALERTA QUE SE GUARDO-->
        </div><!--MENSAJE DE ALERTA QUE SE GUARDO-->
    {% endfor %}<!--MENSAJE DE ALERTA QUE SE GUARDO-->
    {% endif %}<!--MENSAJE DE ALERTA QUE SE GUARDO-->

    {{ register_form.errors }}<!--
aquí muestra los errores del formulario -->

    <form method="POST" action=""><!--
esto se agrega para el formulario de registro-->
        {% csrf_token %}<!--
SIEMPRE AL MANDAR POR POST TIENE QUE IR CSRF_TOKEN -->

        <label for="username">Nombre de usuario</label>
        <input type="text" name="username">
        <label for="password">Contraseña</label>
        <input type="password" name="password">

        <input type="submit" value="Aceptar">
    </form>
{% endblock %}
```

Mostrar usuario identificado.

Index.html (dentro de la carpeta mainapp)

```
{% extends 'layouts/layout.html' %}

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

{% block content %}
    <h1>{{title}}</h1>

    {% if messages %}<!--MENSAJE DE ALERTA QUE SE GUARDO-->
        {% for message in messages %}<!--
MENSAJE DE ALERTA QUE SE GUARDO-->
            <div class="alert alert-success"><!--
MENSAJE DE ALERTA QUE SE GUARDO-->
                {{message}}<!--MENSAJE DE ALERTA QUE SE GUARDO-->
            </div><!--MENSAJE DE ALERTA QUE SE GUARDO-->
        {% endfor %}<!--MENSAJE DE ALERTA QUE SE GUARDO-->
    {% endif %}<!--MENSAJE DE ALERTA QUE SE GUARDO-->

    {% if user.mail %}
        <h2>Bienvenido, {{request.user.first_name}} {{request.user.last_name}}</h2><!--Mostrando datos como nombrecompleto, email -->
        <P>
            {{user.email}}

        </P>
    {% else %}
        <h2>Identificate para ver mas cosas </h2>
    {% endif %}
{% endblock %}
```

Layout.html

```
{% load static %} {% comment 'esto es un comentario' %} load static s
e usa para cargar la carpeta STATIC donde estan los CSS y JS{% endcom
ment %}
<!DOCTYPE html>
<html lang="es">

<head>
    <meta charset="utf-8" />
    <title>
        {% block title %}

        {% endblock %}
        Monchis
    </title>
    <link rel="stylesheet" type="text/css" href="{% static 'css/style
s.css' %}" />
</head>

<body>
    <header>
        <div id="logotipo">
            
            <h1>Sitio web con Django</h1>
        </div>

    </header>

    <nav>
        <ul>
            <li>
                <a href="{% url 'index' %}">Inicio</a> {% comment %}U
sando URL para las rutas{% endcomment %}
            </li>
            {% if user.email %}<!-- Si esta logueado mostrar las demás
opciones
                {% for page in pages %}
                <li>
```

```

        <a href="{% url 'page' page.2 %}">{{page.1}}</a><
!--En context_processors.py 0=id, 1=title, 2=slug -->
        <ul>
            {% for category in categories %}
            <li>
                <a href="{% url 'category' category.0 %}"
>{{category.1}}</a>
            </li>
            {% endfor %}
        </ul>
    </li>
    {% endfor %}
    <li>
        <a href="{% url 'articles' %}">Categorias</a> {%
comment %}Usando URL para las rutas{% endcomment %}
    </li>
    {% else %}
    <li>
        <a href="{% url 'register_page' %}">Registro</a> {% c
omment %}Usando URL para las rutas{% endcomment %}
    </li>
    <li>
        <a href="{% url 'login_page' %}">Login</a> {% comment
%}Usando URL para las rutas{% endcomment %}
    </li>
    {% endif %}←- aquí termina el IF del logueo
    </ul>
</nav>

<div id="content">

    {% block content %}
        <strong>Contenido del bloque original !!!</strong>
    {% endblock %}
</div>

</body>

</html>

```

Cerrar sesión y restringir el acceso.

Views.py (dentro de la carpeta mainapp)

```
def logout_user(request):  
    logout(request)  
    return redirect('login')
```

urls.py (dentro de la carpeta del proyecto principal)

```
path('login_page/', views.login_page, name="login_page"),  
path('logout_user/', views.logout_user, name="logout_user"),
```

Para restringir el acceso a módulos en django se utiliza DECORADORES.

Views.py (dentro de la carpeta mainapp)

```
from django.shortcuts import render, redirect  
from django.contrib import messages # mensaje flash en el registro  
from django.contrib.auth.forms import UserCreationForm # Se importa esto para el formulario de registro de usuarios  
from mainapp.forms import RegisterForm # Se importa de forms.py  
from django.contrib.auth import authenticate, login, logout # Importamos estos módulos para el FORMULARIO DE LOGIN  
from django.contrib.auth.decorators import login_required # Para usar decoradores y restringir el acceso a módulos
```

views.py (dentro de la carpeta blog)

```
# Create your views here.

@login_required(login_url="login_page") ← aquí se usa el DECORADOR y
se restringe el acceso y lo redirecciona a la vista login_page
def articles(request):
    # Sacar articulos
    articles = Article.objects.all()

    #Paginar los articulos
    paginator = Paginator(articles,2) # El numero 2 es cuantos articu
los mostrara por pagina

    #Recoger numero pagina
    page = request.GET.get('page')
    page_articles = paginator.get_page(page)

    return render(request,'articles/list.html',{
        'title': 'Articulos',
        'articles': page_articles
    })
```


Llenar Choices de manera dinámica a partir de una tabla de la base de datos.

Combos simples como por ejemplo: municipios, estados, países.

forms.py

```
class AgregarproductoForm(forms.Form):
    Nombrecorto = forms.CharField()
    Descripcion = forms.CharField()
    Categoria = forms.ModelChoiceField(
        queryset=Categoria.objects.all(), # SE LLENA EL Se
lect combo sencillo de CATEGORIAS dinamicamente con e
l modelo Giro 11/12/2020
        label='Categoria',
        widget=forms.Select
    )
```

Llenar Choices de manera dinámica a partir de una tabla de la base de datos.

Combos multiples es decir seleccion multiple como giros o clasificacion.

forms.py

```
giro_choices = [[g.id, g.descripcion] for g in Giro.objects.all()] # SE LLENA EL SelectMultiple de GIRO dinamicamente con el modelo Giro 11/12/2020
```

```
class AgregarempresaForm(forms.Form):
    nombre = forms.CharField()
    descripcion = forms.CharField()
    domicilio = forms.CharField()
    referencia = forms.CharField()
    telefono = forms.CharField()
    email = forms.CharField()
    logo = forms.ImageField()
    urlpaginaweb = forms.CharField()
    estado = forms.CharField()
    municipio = forms.CharField()
    giros = forms.MultipleChoiceField(choices=giro_choices, widget=forms.SelectMultiple(), required=False) ← aquí se usa el giro_choices
```

Guardar datos ManytoMany en tablas relacionadas por.

Combos multiples es decir seleccion multiple como giros o clasificacion.

views.py

```
def datosempresa(request,id):
    context = {}
    if request.method == "POST":
        form = AgregarempresaForm(request.POST, request.FILES)
        if form.is_valid():
            nombre = form.cleaned_data.get("nombre")
            descripcion = form.cleaned_data.get("descripcion")
            domicilio = form.cleaned_data.get("domicilio")
            referencia = form.cleaned_data.get("referencia")
            telefono = form.cleaned_data.get("telefono")
            email = form.cleaned_data.get("email")
            logo = form.cleaned_data.get("logo")
            urlpaginaweb = form.cleaned_data.get("urlpaginaweb")
            estado = form.cleaned_data.get("estado")
            municipio = form.cleaned_data.get("municipio")
            giros = form.cleaned_data.get("giros")
            creado_por_id = request.user.id
            obj = Empresa.objects.create(
                                nombre = nombre,
```

```

        descripcion = descripcion,
        domicilio = domicilio,
        referencia = referencia,
        telefono = telefono,
        email = email,
        logo = logo,
        urlpaginaweb = urlpaginaweb,
        estado = estado,
        municipio = municipio,
        createdo_por_id = createdo_por_id

        #giros = giros
    )

```

for giro in giros: # Guardando datos ManyToMany en los modelos cuando se tiene un campo Select Multiple 11/12/2020

obj.giro.add(giro) # Guardando datos ManyToMany en los modelos cuando se tiene un campo Select Multiple obj=Empresa,giro= a la tabla empresa_giro 11/12/2020

```

        messages.success(request,'Los datos se guardaron correctamente correctamente.')

```

```
        return redirect('misproductos')
    else:
        form = AgregarempresaForm()
        context['form'] = form
        return render(request, "datos_empresa.html", context)
```

Modificar y eliminar datos ManytoMany en tablas relacionadas por.

Combos multiples es decir seleccion multiple como giros o clasificación hacemos un form en forms.py

views.py

```
from django.db import connection
```

```
@login_required(login_url="login_page")
def modificarempresa(request,id):
    giro_id=0
    idempresa=id
    empresa = get_object_or_404(Empresa, id=id)
    if request.method == "POST":
        form = ModificarempresaForm(request.POST, request.FILES, instance=empresa)
        if form.is_valid():
            #producto = form.save(commit=False)
            giros = request.POST.getlist('giros')
            print(giros)
            messages.success(request,'La empresa se modifico correctamente.')
            empresa.save()
```

```

        with connection.cursor() as cursor: ←AQUÍ
estamos eliminando los datos en las tablas
relacionadas
            cursor.execute("delete from app_empresa_giro where empresa_id="+str(idempresa)) ←AQUÍ
estamos eliminando los datos en las tablas
relacionadas
            row = cursor.fetchone() ←AQUÍ estamos
eliminando los datos en las tablas relacionadas
            for giro in giros: # Guardando datos Many
ToMany en los modelos cuando se tiene un campo Select
Multiple 11/12/2020
                empresa.giro.add(giro)
            return redirect('miempresa',id=request.user.id)
        else:
            giros = Giro.objects.all()
            form = ModificarempresaForm(instance=empresa)
            return render(request, 'modificarempresa.html', {
'form': form, 'giros':giros})

```

forms.py

```

class ModificarempresaForm(forms.ModelForm):
    class Meta:
        model = Empresa
        fields = ['nombre', 'descripcion', 'domicilio',
'referencia', 'telefono', 'email', 'logo', 'urlpaginaweb',
'estado', 'municipio']

```

Enviando parámetros con el redirect:

Views.py

```
return redirect('miempresa',id=request.user.id)
```


Cambio de contraseña con forms :

Views.py

```
from django.contrib.auth import update_session_auth_hash # Para cambiar contraseña con forms
from django.contrib.auth.forms import PasswordChangeForm # Para cambiar contraseña con forms
```

```
def change_password(request):
    if request.method == 'POST':
        form = PasswordChangeForm(request.user, request.POST)
        if form.is_valid():
            user = form.save()
            update_session_auth_hash(request, user)
            # Important!
            messages.success(request, 'Tu password ha sido actualizado correctamente!')
            return redirect('change_password')
        else:
            messages.error(request, 'Porfavor corrija los siguientes errores.')
    else:
        form = PasswordChangeForm(request.user)
        return render(request, 'change_password.html', {
            'form': form
        })
```

urls.py

```
path('change_password/', views.change_password, name="change_password"),
```

change_password.html

```
{% extends 'layouts/layout.html' %}
{% block title %} Agregar producto | Ventas Locales {
% endblock %}
{% block content %}
<!--banner-->
<div class="banner-top">
    <div class="container">
        <h5><a href="{% url 'panel' %}">Inicio</a></l
abel>/Cambiar contraseña</h5>
        <div class="clearfix"> </div>
    </div>
</div>

{% if messages %}
{% for message in messages %}
<div class="alert alert-success" role="alert">
    {{message}}
</div>
{% endfor %}
{% endif %}

<!-- contact -->
<div class="container">
    <div class="row">
```

```

    <!-- left column -->
    <div class="col-md-12">
        <div class="card card-primary">
            <div class="card-header">
                <h3 class="card-
title">Todos los datos son requeridos para guardar el
registro</h3>
            </div>
            <!-- /.card-header -->
            <!-- form start -->
            <form method="POST" action="">
                {% csrf_token %}
                <br><br>
                {{form}}
                <br><br>
                <input class="col-3 btn btn-
block btn-success btn-
sm" type="submit" value="Guardar"><br><br>
            </form>
        </div>
    <!-- /.card -->
    </div>
</div>
{% endblock %}

```

Bibliografia

<https://www.udemy.com/course/master-en-python-aprender-python-django-flask-y-tkinter/learn/lecture/19251224#overview>