**PROYECTO REPOSTERIA DJANGO (guía de proyecto)**

**Quede video 4 1:10:00 minutos**

**Carlos Alfonso Junco Cárdenas**

**Python -m pip install django-widget-tweaks (nos permite usar los widget-tweaks en django )**

**Python –m pip install django-bootstrap4 (nos permite usar bootstrap4 en el proyecto) o podemos abreviarlo de esta forma (pip install django-bootstrap4 )**

**NOTAS RÁPIDAS**

**manage.py runserver (correr el servidor)**

**manage.py makemigrations(prepara los modelos para realizar las migraciones)**

**manage.py migrate(crea las migraciones a la base de datos)**

**manage.py collecstatic (actualiza los archivos estáticos de la aplicación)**

**makemigrations se encarga de crear el archivo para preparar los modelos para la migración, de igual manera si hacemos cambios en nuestros modelos, makemigrations se encarga de actualizar esos cambios**

**con migrate lo que hacemos es aplicar las migraciones a nuestro proyecto, el servidor donde tenemos instalada nuestra base de datos**

**para poder referenciar url, archivos con django dentro de las plantillas html, usamos las etiquetas {% %},**

**por ejemplo “{% url ‘name\_template ’%}” ( como referenciar una url dentro de las las plantillas, el name\_template, es el nombre que usamos para referenciar las url, para no tener que escribir toda la ruta)**

**bootstrap4 nos permite usar utilidades dentro de nuestra aplicación de manera intuitiva, por ejemplo cuando en los formularios usamos la clase form-group, le damos un estilo automático al formulario.**

**los elementos extras que vayamos a usar dentro de la creación de nuestro poryecto lo hacemos dentro del ambiente virtual , que precisamente tendrá un ambiente con las herramientas necesarias para un proyecto especifico**

**cuando creamos una función y no sabemos el número de parámetros que vamos a usar, podemos usar \*args, de esta manera le decimos a la función que puede tomar un número de parámetros indeterminados y además que estos pueden ser de forma iterable, tipo tuplas, listas etc. De manera análoga podemos usar el \*\*kwargs cuando queremos usar varios parámetros, pero estos tienen la característica de tener asociado un clave:valor, es decir para parámetros que sean tipo diccionarios.**

**El método self, se usa en Python cuando queremos acceder a un atributo de una clase, pero desde otro método(función ) o desde otra clase, en pocas palabras cuando queremos referenciar y/o usar un atributo de clase fuera de ella misma.**

**Este documento se crea para suplir la necesidad de ahorrar más de 8 horas de video, enfocado en el proyecto de repostería.**

Nota súper importante: Cada vez que creamos un ambiente virtual nuevo, para proyectos nuevos, es necesario instalar los componentes correspondientes, es decir los elementos que están en uso en otro ambiente no servirán para el nuevo, ya que un ambiente virtual precisamente lo que hace es aislar cada proyecto de otros proyectos en Django y del lenguaje nativo de Python, para evitar conflictos entre aplicaciones y Python nativo. En el punto número 11, podemos buscar algunas tips de elemento dentro de un htl

Nota importante sobre forms: podemos usar los forms de django con nuestros modelos, de esta manera importamos la clase forms de django, o podemos crear nuestros forms.py aparte y luego asociarlo a las clases

Esta última opción me parece más ideal porque podemos controlar el formato de entrada en nuestros formularios, para esto se usan los widgets.

**Antes de empezar recordemos que bootstrap es una librería que me permite usar archivos css, js de manera más intuitiva y más fácil, podemos descargar bootstrap desde la página oficial. Jquery análogamente es otra librería rica en funciones JavaScript que nos permite poder implementar animaciones, manejar eventos en las páginas web.**

**Cuando trabajamos en un proyecto donde se requiere tener mayor ahorro de tiempo, puede ser bastante provechoso reciclar código e inclusive usar plantillas, principalmente para el desarrollo de las templates de nuestro proyecto, por ejemplo, cuando creamos una barra de navegación.**

**Utilizaremos el Framework Django para Python para crear nuestro proyecto, Un framework es una herramienta de desarrollo web que, por lo general, se define como una aplicación o conjunto de módulos que permiten el desarrollo ágil de aplicaciones mediante la aportación de librerías y/o funcionalidades ya creadas (1).**

Instalamos Django en nuestro computador, en la raíz del disco duro, si usamos Windows 10, nos vamos a la raíz con el comando cd/, también podemos instalar django en la carpeta donde guardamos nuestro proyecto

Python -m pip install django (para la versión en mac, OS se usa el sudo easy\_install django)

creamos una carpeta donde instalamos un ambiente virtual, un ambiente virtual es una especie de contenedor que nos permite, aislar los proyectos con dependencias para que no interfiera con Python nativo del sistema, es decir crea un ambiente aislado, donde se dedica a usar librerías y recursos específicos sólo para el proyecto, se recomienda crear un ambiente virtual para cada proyecto, colocamos el comando, lo podemos instalar en la raíz de nuestro disco Python -m pip virtaulenv

Luego creamos un ambiente virtual Python -m venv ambiente\_reposteria (nombre del ambiente virtual)

Luego instalamos django en nuestro ambiente virtual, para asegurarnos de que sea funcional en nuestro ambiente independizado

(ambiente\_reposteria) C:\ambientes\ambiente\_reposteria>python -m pip install django

1) ya que tenemos instalado django en nuestro disco duro y nuestro entorno virtual creado, procedemos a crear nuestro proyecto que será la base para el desarrollo web.

Activamos el ambiente virtual con C:\ambientes\ambiente\_reposteria\Scripts>activate, nos movemos en las carpetas con el comando cd, cd.. para regresar al directorio anterior y cd/ para regresar a la raíz. Luego ingresamos hasta la carpeta donde está nuestro proyecto, y colocamos el siguiente comando (ojo recuerda que antes tenemos que salir hasta la raíz).

(ambiente\_reposteria) C:\proyectosDjango>django-admin.py startproject resposteria

2) Creamos una carpeta dentro del proyecto repostería, que se llamará apps, en esta creamos

un archivo \_\_init\_\_.py (que le dice a django que donde esté este archivo es un paquete de Python)

3) Estando ubicados en la carpeta apps creamos nuestra primera aplicación

(ambiente\_reposteria) C:\proyectosDjango\reposteria\apps>django-admin.py startapp postres

3.1) Instalamos el cliente para poder asociar nuestro proyecto con mysql (ambiente\_reposteria) C:\proyectosDjango\pdreposteria>python -m pip install mysqlclient

4) Ahora procedemos a empezar a modificar nuestra aplicación, para esto podemos ir al archivo settings de proyecto y registrar inicialmente la aplicación. Realizar unas configuraciones básicas para nuestra base de datos (para este caso será, mysql)

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'apps.postres',

]

Configuración de la base de datos

Las letras verdes con el signo #, son comentarios dentro del framework.

DATABASES = {

    'sqlite': {

        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

    },

      'default': {

        'ENGINE': 'django.db.backends.mysql',# motor de la base de datos, define que base de datos usar

        'NAME':'bdproductos',

        'HOST': 'localhost', #anfitrion que me permite usar el servicio web( van a partir desde este computador, terminan en este computador)

        'PORT':'3306', # interfaz que permite realizar peticiones o recibirlas a través del servidor

        'PASSWORD':'',

        'USER':'root', # modo que me permite tener permisos de administrador

        'OPTIONS':{

          'sql\_mode':'traditional',

        }

    },

}

5) nos vamos a los modelos y creamos una clase postres con sus atributos, para poder instanciar a partir de ella (instanciar es crear objetos que para este caso serías los registros a partir de una clase,)

Se importa la librería timezone para poder tener registro de cuando un elemento se crea

from django.db import models

from django.utils import timezone

# Create your models here.

class Postres(models.Model):

  nombre = models.CharField(max\_length=100, default='DEFAULT VALUE')

  precio = models.CharField(max\_length=20, default='DEFAULT VALUE')

  stock =  models.CharField(max\_length=100, default='DEFAULT VALUE')

  img = models.CharField(max\_length=100, default='DEFAULT VALUE')

  created\_at = models.DateTimeField(auto\_now\_add=True) # DateTime me permite guardar la fecha de creación

de un registro asociado

  update\_at = models.DateTimeField(auto\_now=True)

# define un objeto a la base de datos, propiedad de la base de datos, define metadato sobre la clase creada

class Meta:

    db\_table='postres'

Los metadatos sirven para describir información de los datos

5.1) Hacemos una primera migración para probar nuestro modelo, teniendo el ambiente virtual activo, recordemos que el ambiente virtual, nos permite independizar nuestro proyecto con características aparte de otros proyectos.

(ambiente\_reposteria) C:\proyectosDjango\reposteria>manage.py makemigrations

(ambiente\_reposteria) C:\proyectosDjango\reposteria>manage.py migrate

6) Creamos las vistas para el modelo de postres, que nos permitirá asociar el crud (CREATE, READ(LIST), UPDATE, DELETE)

from django.shortcuts import render

from django.views.generic import ListView,DetailView

#se usan las vistas que nos permitirá manipular los datos

from django.views.generic.edit import CreateView,UpdateView,DeleteView

# importamos los modelos para poder usarlos en nuestras vistas;

from apps.postres.models import Postres

# vamos a crear las vistas asociadas al crud, de tal manera que las vistas deciden que mostrar en la aplicación web

# las plantillas definen como se muestran las vistar, todo el aspecto visual.

# redireccionamiento de las url

from django.urls import reverse

from django.urls import reverse\_lazy

#me permite usar mensajes dentro de las clases, por ejemplo cuando se instancia un objeto de una clase

from django.contrib import messages

# permita trabar con los mensajes a través de las vistas

from django.contrib.messages.views import SuccessMessageMixin

# habilitamos los formularios genérico  en django, es decir los formularios

from django import forms

# vamos a listar los elementos de la clase postres

class PostreListado(ListView):

    model=Postres

#se pone mensajes para que me notifique cuando se cree un registro

class PostreCrear(SuccessMessageMixin,CreateView):

    model=Postres

    form=Postres

    fields='\_\_all\_\_'

    success\_message='postre creado de manera exitosa'

# redirecciona a la página principal después de crear un postre

def get\_success\_url(self):

    return reverse('leer')

# vamos a crear una clase que me detalle los postres

class PostresDetalle(DetailView):

    model=Postres

class PostresActualizar(UpdateView,SuccessMessageMixin):

    model=Postres

    form=Postres

    fields='\_\_all\_\_'

    success\_message='postre actualizado con éxito'

    def get\_success\_url(self):

        return reverse('leer')

class PostresEliminar(DeleteView,SuccessMessageMixin):

    model=Postres

    form=Postres

    fields='\_\_all\_\_'

    # las funciones me permiten redireccionar a la página principal

    def get\_success\_url(self):

       success\_message='postre eliminado con éxito'

       messages.success=(self.request,(success\_message))

       return reverse('leer')

self se utiliza en las funciones de Python para hacer referencia al objeto instanciado de una clase, PostresListado y PostresDetalles, no requieren retornar con la url a otra página pues estás no mostraron datos (listar y observar detalles de los registros creados);

7)Ahora vamos a crear las rutas en el archivo urls general del proyecto, podemos hacerlo con cada aplicación individual, pero en este caso las haremos en url.py de repostería y no para cada aplicación (nos vamos al archivo url.py de repostería)

Cuando creamos las rutas, colocamos el nombre de la aplicación relacionado con la ruta seguido con el nombre de la vista

Por ejemplo apps Postres, nombre de vista PostresCrear

Path(“postres/crear”)…

from django.contrib import admin

from django.urls import path

from apps.postres.views import PostreListado,PostreCrear,PostresActualizar,PostresDetalle,PostresEliminar

urlpatterns = [

    path('admin/', admin.site.urls),

    #ruta para listar todos los postres

    path('postres',PostreListado.as\_view(template\_name='postres/index.html'), name='leer'),

    path('postres/detalle/<int:pk>',PostresDetalle.as\_view(template\_name='postres/detalles.html'), name='detalles'),

    path('postres/crear/',PostreCrear.as\_view(template\_name='postres/crear.html'), name='crear'),

    path('postres/editar/<int:pk>',PostresActualizar.as\_view(template\_name='postres/editar.html'),name='atualizar'),

    path('postres/eliminar/<int:pk>',PostresEliminar.as\_view(), name='eliminar'),

]

A PostresEliminar no le ponemos nada como argumento, pues al eliminar simplemente deja de existir el registro

8) Ahora creamos una carpeta en nuestro proyecto que se va a llamar templates, donde crearemos las plantillas para nuestro proyecto, podemos usar el comando mk dir templates, o de otra manera usamos el md templates, o creamos la carpeta directamente.

(ambiente\_reposteria) C:\proyectosDjango\reposteria>md templates

Dentro de templates, creamos una carpeta base, donde crearemos nuestra base para todo el desarrollo (recordemos que la plantilla se encarga de la parte visual de cómo mostrar las vistas, en el modelo, vista plantilla)

(ambiente\_reposteria) C:\proyectosDjango\reposteria\templates>mkdir base

9) vamos a crear nuestro archivo base para el proyecto

Framework, marco de trabajo que permite organizar el código y realizar funcionalidades de manera más optimizada

En nuestro archivo html base, necesitamos referenciar los estilos que vamos a usar y los javascript que necesitaremos, recordemos que bootstrap es una librería que nos aporta funciones CSS, Javascript.

se referencia dentro de estas etiquetas {%%}

{% load static%}

<!DOCTYPE html>

<html lang="es">

<head>

    {% block extrahead %}

    {% load bootstrap4 %}

    {% load static %}

    {% bootstrap\_css %}

    {% load widget\_tweaks %}

    <link rel="stylesheet" type="text/css" href="{% static 'css/estilos.css' %}">

    {% endblock %}

    <meta charset="UTF-8">

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

    <link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">

    <link rel="stylesheet" href="{% static 'css/bootstrap.css' %}">

    <title>{% block title %} Document {% endblock %}</title>

</head>

<body>

<div class="content container">

        <div class="row">

            <div class="col-md-10">

             <br><br><br><br>

            </div>

        </div>

</div>

    {% block navbar %}

    <div class="navbar navbar-expand-lg fixed-top navbar-dark bg-primary">

      <div class="container">

        <a href="../" class="navbar-brand">Inicio</a>

        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarResponsive" aria-controls="navbarResponsive" aria-expanded="false" aria-label="Toggle navigation">

          <span class="navbar-toggler-icon"></span>

        </button>

        <div class="collapse navbar-collapse" id="navbarResponsive">

          <ul class="navbar-nav">

            <li class="nav-item dropdown">

              <a class="nav-link dropdown-toggle" data-toggle="dropdown" href="#" id="themes">Postres <span class="caret"></span></a>

              <div class="dropdown-menu" aria-labelledby="themes">

                <a class="dropdown-item" href="{% url 'leer' %}">Registrar</a>

                <div class="dropdown-divider"></div>

                <a class="dropdown-item" href="{% url 'leer' %}">Listar</a>

              </div>

            </li>

              <li class="nav-item dropdown">

                <a class="nav-link dropdown-toggle" data-toggle="dropdown" href="#" id="themes">Clientes <span class="caret"></span></a>

                <div class="dropdown-menu" aria-labelledby="themes">

                  <a class="dropdown-item" href="{% url 'leer' %}">Solicitar</a>

                  <div class="dropdown-divider"></div>

                  <a class="dropdown-item" href="{% url 'leer' %}">Listar Solicitud</a>

                </div>

              </li>

          </ul>

        </div>

        <ul class=" nav navbar-nav navbar-right">

          <li class="nav-item dropdown">

            <a class="nav-link dropdown-toggle" data-toggle="dropdown" href="#" id="themes">Opciones <span class="caret"></span></a>

            <div class="dropdown-menu" aria-labelledby="themes">

              <a class="dropdown-item" href="{% url 'leer' %}">Cerrar sesión</a>

            </div>

          </li>

            <li class="nav-item dropdown">

              <a class="nav-link dropdown-toggle" data-toggle="dropdown" href="#" id="themes">Adopciones <span class="caret"></span></a>

              <div class="dropdown-menu" aria-labelledby="themes">

                <a class="dropdown-item" href="{% url 'leer' %}">Solicitar</a>

                <div class="dropdown-divider"></div>

                <a class="dropdown-item" href="{% url 'leer' %}">Listar Solicitud</a>

              </div>

            </li>

        </ul>

      </div>

    </div>

{% endblock %}

<div class="content container">

    <div class="row">

        <div class="col-md-10">

         {% block content %}

         {% endblock %}

        </div>

    </div>

</div>

{% bootstrap\_javascript jquery='full' %}

<script src="{% static 'js/jquery.min.js' %}"></script>

<script src="{% static 'js/bootstrap.min.js' %}"></script>

</body>

</html>

10) Creamos dentro de templates otra carpeta que se llame postres, donde vamos a crear todos los archivos asociados a la manipulación, creación de los registros de postres (crear, eliminar, modificar, listar), creamos index.html, crear.html, detalles.html.

Empezamos creando la plantilla para listar los postres, heredando de la plantilla base la visualización general, lo que haremos es sobrescribir ciertos elementos dentro de la plantilla y adicionar otros elementos.

{% extends 'base/base.html' %}

{% load static %}

{% block content %}

<div class="contanier text-center mt-5">

    <div class="row">

        <div class="col-md-12">

        <br><br>

        <h5>Lista de postres</h5>

        </div>

    </div>

    <div class="row mb-5">

        <div class="card col-md-2">

            <h6 class="mt-4"><strong>Menú</strong></h6>

            <ul class="list-group list-group-flush">

                <li class="list-group-item">Clientes</li>

                <li class="list-group-item active">Postres</li>

                <li class="list-group-item">Pedido</li>

            </ul>

        </div>

        <div class="col-md-10">

            <div class="table-responsive">

            {% if messages %}

            <ul class="messages list-group mb-3">

                 {% for message in messages %}

                 <li {% if message.tags %} class="{{ message.tags}} list-group-item list-group-item-primary"{% endif %}>{{ message }}</li>

                 {% endfor %}

             </ul>

            {% endif %}

                <div align="left" class="btn\_crear mb-3">

                    <a href="crear" type="button" class="btn-btn-primary">crear</a>

                </div>

                <table class="table table-striped table-hover">

                <thead>

                    <tr>

                        <th width="35%%">Nombre</th>

                        <th>Precio</th>

                        <th>Stock</th>

                        <th>Imagen</th>

                        <th>Acciones</th>

                    </tr>

                </thead>

                <tbody>

                    {% if object\_list %}

                    {% for postre in object\_list  %}

                    <tr>

                        <td>{{ postre.nombre }}</td>

                        <td>{{ postre.precio }}</td>

                        <td>{{ postre.stock }}</td>

                        <td><img src="{% static 'uploads/' %}{{postre.img}}" alt="{{postre.img}}" class="img-fluid" width="7%"></td>

                        <td>

                        <form method="POST" action="eliminar/{{postre.id}}">

                          {% csrf\_token %}

                            <div class="btn-group">

                                <a href="detalle/{{postre.id}}" title="Ver" class="btn btn-success">Ver</a>

                                <a href="editar/{{postre.id}}" title="Editar" class="btn btn-primary">Editar</a>

                                    <button class="btn btn-danger" onclick="return eliminar();" type="submit">

                                    Eliminar

                                    </button>

                            </div>

                        </form>

                        </td>

                    </tr>

                    {% endfor  %}

                    {% else  %}

                    <h1>No hay registro de postres</h1>

                    {% endif %}

                <tbody>

            </table>

            </div>

        </div>

    </div>

</div>

<script type="text/javascript">

    function eliminar() {

        var x = confirm("Eliminar producto?");

        if (x)

            return true;

        else

            return false;

    }

</script>

{% endblock %}

El método post me permite realizar solicitudes de modificación, submit nos permite hacer solicitudes de envío a tráves de un botón, cuando creamos un grupo de elementos, podemos usar group-item para referenciar que hace parte del grupo

11) Ahora vamos a crear nuestra plantilla que nos permita crear nuevos registros de postres, el elemento main, nos permite mostrar, o acentuar el contenido principal en un body (cuerpo) del html., responsive es un tipo de diseño web que nos permite adaptar los elemento a distintos tipos de dispositivos. La etiqueta nav me crea una barra de navegación.

los scripts los cargamos dentro de una etiqueta script, src sirve para referenciar las rutas del archivo JavaScript que pueden ir dentro del cuerpo, los archivos de estilo CSS, los podemos referenciar dentro de link en la cabecera, href sirve para contener la ruta del archivo CSS, o JavaScript a usar.

rel=”stylesheet" describe la relación para poder enlazar el archivo que traemos aparte dentro del html, como una hoja de estilo

 bootstrap nos permite implementar estilos CSS y funciones javascript dentro de nuestro desarrollo web, de una manera más sencilla. Sin embargo podemos usar archivos CSS y JavaScript para adicionar funciones y estilos a nuestro proyecto.

La línea (./) sirve en los vínculos para regresar a la página anterior

Creamos ahora la plantilla para poder hacer registros de postres, crear registros

{% extends 'base/base.html '%}

{% block content %}

{% comment %}  el main define el

elemento principal del cuerpo del trabajo {% endcomment %}

<main role='main'>

  <div class='row mb-5'>

    <div class='col-md-10'>

      <div class='table responsive'>

      {% comment %} podemos definir como se pueden ver nuestras plantillas, adicionando los elementos que queremos,

      en este caso cuando creamos un postre, nos saldrá una barra de navegación {% endcomment %}

         <nav aria-label=<nav class="breadcrumb mb-3">

          {% comment %} para este caso creamos una barra donde se encuentren las opciones de postres, crear {% endcomment %}

            <ol class='breadcrumb'>

                <li class="breadcrumb-item"><a href="./">Postres</a></li>

                <li class="breadcrumb-item active">Crear</li>

            </ol>

         </nav>

         {% comment %} aquí creamos el formulario para poder ingresar los datos de nuestro registro {% endcomment %}

        <div class='form align-left'>

           <h4>Crear</h4>

              {% comment %} enctype nos permite subir archivos al servidor {% endcomment %}

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

                  {% csrf\_token %}

                  <div class='form-group'>

                    <label form='nombre' class='text\_negrita'>Nombre</label>

                    {{ form.nombre }}

                  </div>

                   <div class='form-group'>

                    <label form='precio' class='text\_negrita'>Precio</label>

                    {{ form.precio}}

                  </div>

                  <div class='form-group'>

                    <label form='stock' class='text\_negrita'>Stock</label>

                    {{ form.stock}}

                  </div>

                  <div class='form-group'>

                    <label form='img' class='text\_negrita'>Imagen</label>

                    {{ form.img}}

                  </div>

                  {% comment %} no se coloca el cretead\_at y update\_at por que se crean automaticamente {% endcomment %}

                  <button type='submit' class='btn btn-primary'>Aceptar</button>

{% comment %} colocamos una etiqueta que me permita cancelar la acción  {% endcomment %}

                  <a href='./' type='submit' class='btn btn-primary'>Cancelar</a>

             </form>

        </div>

      </div>

    </div>

  </div>

</main>

{% endblock %}

12) en settings py en installed apps agregamos bootstrap4 y widget-tweaks que cargamos en la plantilla base, en el mismo settings modificamos las url para los archivos estáticos de tal manera que se puedan mostrar los archivos que estamos cargando.

STATIC\_URL = '/static/'

STATICFILES\_DIRS= (os.path.join(BASE\_DIR, 'static')

MEDIA\_URL = '/postres/'

MEDIA\_ROOT= os.path.join(BASE\_DIR, 'static/uploads')

MESSAGE\_STORAGE = 'django.contrib.messages.storage.cookie.CookieStorage'

13) miramos el directorio de las templates

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [os.path.join(BASE\_DIR, 'templates')],

        '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',

            ],

        },

    },

]

Adicionamos dentro de las aplicaciones bootstrap4 y widget-tweaks que es una herramienta para poder manipular los formatos de los formularios.

14)

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'apps.postres',

    'bootstrap4',

    'widget\_tweaks',

]

Ahora vamos a modificar el modelo de postres para agregar un control en estado (activo-inactivo), para esto vamos a crear una tupla dentro de una lista.

15)

Aquí nueva modificación en archivo models.py en la aplicación postres

from django.db import models

from django.utils import timezone

# Create your models here.

# vamos a crear una lista, con una tupla interna

postres\_estado=[

    (1,'Activo'),

    (2,'Inactivo'),

]

# vamos a intentar crear otra forma de cambiar de estado

# si  no colocamos la primary key dentro

class Postres(models.Model):

    nombre=models.CharField(max\_length=100)

    precio=models.FloatField()

    stock=models.IntegerField()

    #me permite adicionar archivos

    img=models.FileField()

    estado=models.IntegerField(null=False, blank=False,choices=postres\_estado, default=1)

    created\_at=models.DateTimeField(auto\_now\_add=True)

    update\_at=models.DateTimeField(auto\_now=True)

class Meta:

    db\_table='postres'

luego adicionar en la template de crear, el campo que nos permita ingresar el registro de cambio estado.

Colocar en (template/postres/crear)

       <div class="form-group">

                                <label for="estado" class="txt-negrita">Estado</label>

                                {{form.estado}}

                            </div>

16) ahora necesitamos crear nuestra plantilla de detalles para poder visualizar el detalle de los postres registrados, creamos un archivo dentro de la carpeta template que se llamará detalles.html

La etiqueta span, nos sirve para agrupar elementos en línea, o darle estilo al texto

{% extends 'base/base.html' %}

{% load static %}

{% block content %}

    <main role="main">

        <div class="text-center mt-5">

            <br><br>

            <div class="row mb-4">

                <div class="col md-12">

                    <h1 style="font-size: 28px;">Postres</h1>

                </div>

            </div>

            <div class="row">

                <div class="col md-10">

                    <div class="table-responsive">

                        <nav aria-label="breadcrumb">

                            <ol class="breadcrumb">

                                <li class="breadcrumb-item"><a href="./">Postres</a></li>

                                <li class="breadcrumb-item activate">Detalles</li>

                            </ol>

                        </nav>

                        <div class="form align.left">

                            <h4>Detalles de postres</h4>

                            <p><span class="txt\_negrita">Nombre:</span><br>{{ object.nombre}}</p>

                            <p><span class="txt\_negrita">Precio:</span><br>{{object.precio}}</p>

                            <p><span class="txt\_negrita">Stock:</span><br>{{object.stock}}</p>

                            <p><span class="txt\_negrita">Imagen:</span><br><img src="{% static 'uploads/'%}{{object.img}}" class="img-fluid"></p>

                            <p><span class="txt\_negrita">Creado:</span><br>{{object.created\_at}}</p>

                            <p><span class="txt\_negrita">Actualizado:</span><br>{{object.update\_at}}</p>

                            <p><span class="txt\_negrita">Estado:</span><br>{{object.estado}}</p>

                            <a href="../" type="submit" class="btn btn-primary">Regresar</a>

                        </div>

                    </div>

                </div>

            </div>

        </div>

    </main>

{% endblock %}

17) ahora vamos a crear un archivo actualizar.html donde vamos a actualizar los postres, podemos hacerlo de dos formas, copiando el archivo crear.html , pegarlo en actualizar y modificar algunos aspectos dentro del archivo

Img fluid me permite adaptar la imagen

Dentro de las etiquetas imágenes, y links el href se usa para cargar links que lleven a otro documento, página

Mientras que src me permite cargar archivos dentro del documento.

Otro aspecto importante es que necesitamos de un atributo que nos permite subir archivos con el método post que colocamos dentro de la etquita <form method=”post”>, usamos el atributo enctype=”multipart/form-data”

{% extends 'base/base.html'%}

{% load static %}

{% block content %}

    <main role="main">

        <div class="container text-center mt-5">

            <br><br>

            <div class="row mb-4">

                <div="col md-12">

                    <h1 style="font-size:28px;">Postres </h1>

                </div>

            </div>

            <div class="row mb-5">

                <div class="col md-10">

                    <div class="table-responsive">

                        <nav aria-label="breadcrumb mb-3">

                            <ol class="breadcrumb">

                                <li class="breadcrumb-item">Postres<a href="./"></a></li>

                                <li class="breadcrumb-item">Editar<a href="#"></a></li>

                                <li class="breadcrumb-item active" aria-current="page">{{object.nombre}}</li>

                            </ol>

                        </nav>

                    </div>

                </div>

            </div>

            {% comment %} hacemos el llamado para poder instanciar registros, pero esta vez, los actualizaremos  {% endcomment %}

            <div class="row mb-5">

                <div class="col md-10">

                    <div class="table-responsive">

                        <div class="form align-left">

                        {% comment %} enctype sirve para poder subir archivos dentro de un formulario {% endcomment %}

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

                                {% csrf\_token %}

                                <div class="form-group">

                                    <label for="nombre" class="txt\_negrita">Nombre</label>

                                    {{form.nombre}}

                                </div>

                                <div class="form-group">

                                    <label for="precio" class="txt\_negrita">Precio</label>

                                    {{form.precio}}

                                </div>

                                <div class="form-group">

                                    <label for="stock" class="txt\_negrita">Stock</label>

                                    {{form.stock}}

                                </div>

                                {% comment %} vamos a colocar la imagen actual, y la imagen que queremos actualizar {% endcomment %}

                                <div class="form-group">

                                    <label for="img" class="txt\_negrita">Imagen</label>

                                    {{ form.img }}

                                    <p class="txt\_negrita">Imagen actual</p>

                                    {% comment %} etiqueta para mostrar la imagen actual del registro  {% endcomment %}

                                    <img src="{% static 'uploads/'%}{{ object.img }}" class="img-fluid" alt="{{object.nombre}}">

                                </div>

                                <div class="form-group">

                                    <label for="estado" class="txt-negrita">Estado</label>

                                    {{form.estado}}

                                </div>

                                {% comment %} crearemos los botones para actualizar {% endcomment %}

                                <button type="submit" class="btn btn-primary ">Aceptar</button>

                                {% comment %} enlace que me permita cancelar la operación y me retorno a tarvés del botón a la página anterior {% endcomment %}

                                <a href="../" type="submit" class="btn btn-primary">Volver</a>

                            </form>

                        </div>

                    </div>

                </div>

            </div>

        </div>

    </main>

{% endblock %}

18) ahora vamos a crear el resto de modelos para otras apps, creamos inicialmente la primera aplicación que se llamará clientes

(ambiente\_reposteria) C:\proyectosDjango\reposteria\apps>django-admin.py startapp clientes

De igual manera creamos otra app, que se llamará pedidos (ambiente\_reposteria) C:\proyectosDjango\reposteria\apps>django-admin.py startapp pedido, luego debemos agregar las aplicaciones en el archvio settings, esto para que lo reconozca el proyecto

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'apps.postres',

    'apps.pedidos',

    'apps.clientes',

    'bootstrap4',

    'widget\_tweaks',

]

19) Nos vamos a models, de la app de clientes y creamos una clase llamada Clientes

from django.db import models

# Create your models here.

class Clientes(models.Model):

    id\_clientes=models.CharField(max\_length=50)

    nombre=models.CharField(max\_length=50)

    apellidos=models.CharField(max\_length=70)

    direccion=models.TextField()

    ciudad=models.CharField(max\_length=50)

    telefono=models.IntegerField()

    email=models.EmailField()

# creamos una función ahora para que en caso de tener que concatenar el nombre y el apellido,

# no tengamos errores

def \_\_str\_\_(self):

    return '{} {}'.format(self.nombre, self.apellido)

20) ahora nos vamos a models de pedidos y creamos la clase de pedidos, de la siguiente manera

Ahora vamos a asociar la clase Pedidos con la clase Clientes, recordemos que podemos asociar de manera relacional dos clases,

Un cliente puede generar uno o muchos pedidos, por lo tanto, esta relación es de uno a muchos

Recordemos que, en una relación de uno a muchos, la llave foránea se crea donde la relación cae a muchos

La clase detalle pedidos, es una tabla intermedia para hacer una relación de los pedidos con relación al número de productos pedidos, sobre si aplica un descuento o algo similar.

from django.db import models

from apps.clientes.models import Clientes

from apps.postres.models import Postres

# Create your models here.

# creamos una relación de uno a muchos de la clase cliente con respecto a la clase pedidos, de esta manera

# podemos ver que un cliente puede generar mucho pedidos

class Pedidos(models.Model):

    clientes=models.ForeignKey(Clientes, null=True, blank=True, on\_delete=models.CASCADE)

    fechapedido=models.DateTimeField()

    fechaentrega=models.DateTimeField()

    fechaenvio=models.DateTimeField()

    formaenvio=models.CharField(max\_length=50)

    destinatario=models.CharField(max\_length=70)

    direcciondestinatario=models.TextField()

    ciudaddestinatario=models.CharField(max\_length=70)

    # nos permitirá tener un control sobre los pedidos que se hagan,

    # un postre puede tener varios detalles de pedidos

class DetallePedido(models.Model):

    postre=models.ForeignKey(Postres,null=True, blank=True, on\_delete=models.CASCADE)

    pedidos=models.ForeignKey(Pedidos, null=True, blank=True, on\_delete=models.CASCADE)

    preciounidad=models.FloatField()

    descuento=models.IntegerField()

21) luego podemos proceder a crear las migraciones de los modelos en Pedidos y Clientes

(ambiente\_reposteria) C:\proyectosDjango\reposteria>manage.py makemigrations clientes

(ambiente\_reposteria) C:\proyectosDjango\reposteria>manage.py makemigrations pedidos

Luego hacemos el migrate para poder realizar las migraciones a la base de datos

(ambiente\_reposteria) C:\proyectosDjango\reposteria>manage.py migrate clientes

(ambiente\_reposteria) C:\proyectosDjango\reposteria>manage.py migrate pedidos

22) Ahora vamos a crear dos carpetas una usuario asociado al logeo de usuario y una registration que nos permitirá gestionar el cambio de contraseña

Comenzamos con la carpeta de usuario, dentro de ella creamos el archivo de index.html y registrar.html, el primero lo crearemos para loguearnos a nuestra aplicación y el segundo para registrar un administrador

En el archivo de index.html, vamos a crear los formularios para ingresar nuestro usuario

{% extends 'base/base.html'%}

{% block title %}

{% endblock %}

{% block navbar %}

{% endblock %}

{% block content %}

{% comment %} método post que me permite hacer solicitudes de modificación  {% endcomment %}

<form method="POST">

{% comment %} token de seguridad para el formulario {% endcomment %}

    {% csrf\_token %}

<div class="table table-responsive">

    <div class="row">

        <div class="col-md-6 col-md-offset-3">

            <div class="form-group">

                <label for="username">Nombre de usuario</label>

                <input class="form-control" type="text" name="username">

            </div>

            <div class="form-group">

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

                <input class="form-control" type="password" name="password">

            </div>

            {% comment %} finalmente creamos un botón que nos lleve a una página para reestablecer la contraseña {% endcomment %}

            <div class="form-group">

                <a href="{% url 'password\_reset'%}">Olvidé la contraseña</a>

                <input class=" btn btn-primary" type="submit">

            </div>

        </div>

    </div>

</div>

</form >

{% endblock %}

23)

Hacemos lo mismo con registrar.html para poder crear los formularios para registrar un nuevo administrador

vamos a crear el formulario para poder agregar un nuevo usuario

pero debemos ver que los campos concuerden con los campos de la tabla user.

{% extends 'base/base.html' %}

{% block content %}

<form method="POST">

    {% csrf\_token %}

    <div class="table table-responsive">

        <div class="row">

            <div class="col-md-8 col-md-offset-2">

                <div class="form-group">

                    <label for="first\_name">Nombre</label>

                    <input class="form-control" type="text" name="first\_name">

                </div>

                <div class="form-group">

                    <label for="last\_name">Apellido</label>

                    <input class="form-control" type="text" name="last\_name">

                </div>

                <div class="form-group">

                    <label for="usaername">Nombre de usuario</label>

                    <input class="form-control" type="text" name="username">

                </div>

                <div class="form-group">

                    <label for="email">Correo electrónico</label>

                    <input class="form-control" type="text" name="email">

                </div>

                <div class="form-group">

                    <label for="passwpord1">Contraseña</label>

                    <input class="form-control" type="password" name="password1">

                </div>

                <div class="form-group">

                    <label for="password2">Confirmar contraseña</label>

                    <input class="form-control" type="password" name="password2">

                </div>

                <div class="form-group">

                    <button class="btn btn-primary" type="submit ">Aceptar</button>

                </div>

            </div>

        </div>

    </div>

</form>

{% endblock %}

23) ahora necesitamos crear la aplicación para de usuario para poder asociar las templates con los modelos y las vistas que necesitamos crear para usuario

(ambiente\_reposteria) C:\proyectosDjango\reposteria\apps>django-admin.py startapp usuario

Procedemos a crear la vista para nuestra app de usuario, no creamos modelos para esta aplicación, pues usamos

El modelo user que trae django, por esa razón cuando creamos las templates nos fijamos en los campos que salen en la tabla de user, esos campos son los que usaremos como modelo base para las vistas.

from django.shortcuts import render

from django.contrib.auth.models import User

from django.contrib.auth.forms import UserCreationForm

from django.views.generic import CreateView

from django.urls import reverse\_lazy

from apps.usuario.form import RegistroForm

# Create your views here.

class RegistroUsuario(CreateView):

    model=User

    template\_name='usuario/registrar.html'

    form\_class=RegistroForm

    success\_url=reverse\_lazy('leer')

Creamos un archivo nuevo en usuario, que se llamará form.py , donde colocamos la clase forms.Form, esto nos permitirás personalizar nuestros formularios.

from django.contrib.auth.models import User

from django.contrib.auth.forms import UserCreationForm

class RegistroForm(UserCreationForm):

    class Meta:

        model=User

        fields = [

            'username',

            'first\_name',

            'last\_name',

            'email',

        ]

        label = {

            'username':'Nombre de usuario ',

            'first\_name': 'Nombre',

            'last\_name':'Apellido',

            'email':'correo electrónico',

        }

24) nos vamos a las urls y hacemos las importaciones para el logeo y cambio de contraseña

from django.contrib.auth.views import LoginView,LogoutView,PasswordResetView,PasswordResetDoneView,PasswordResetConfirmView,PasswordResetCompleteView,logout\_then\_login

Vamos a importar las vistas para cambiar la contraseña y para el login, para la url del login de usuario no necesitamos crear una vista para el logeo, ya que importamos la vista de loginview que trae django y colocamos el template\_name asociado al que creamos para el logeo de la siguiente manera.

Dentro de las urlpatterns del archivo urls del proyecto

  path('accounts/login/',LoginView.as\_view(template\_name="usuario/index.html"), name="login"),

ahora creamos la ruta para el usuario nuevo, pero antes debemos importar la vista que creamos para registrar un nuevo usuario, a diferencia del login que usamos una vitsa por defecto de django, y por lo tanto no fue necesario crear una vista

from apps.usuario.views import RegistroUsuario

y creamos la ruta dentro las urlpatterns

path('usuario/nuevo',RegistroUsuario.as\_view(), name="registrar"),

finalmente debemos instalar nuestra aplicación dentro del archivo settings installed apps para que pueda ser reconocida dentro del proyecto

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'apps.postres',

    'apps.pedidos',

    'apps.clientes',

    'apps.usuario',

    'bootstrap4',

    'widget\_tweaks',

]

25) ahora necesitamos hacer las configuraciones para poder hacer el cambio de contraseña dentro de setting escribimos lo siguiente

Importamos el reverse\_lazy para poder usar el redireccionamiento

import os

# importamos el reverse\_lazy

from django.urls import reverse\_lazy

luego seguimos con la configuración

# escribimos la configuración para cambio de contraseña y redirección de la ruta

LOGIN\_REDIRECT\_URL=reverse\_lazy('leer')

LOGIN\_REDIRECT\_URL=reverse\_lazy('login')

EMAIL\_USE\_TLS=True

EMAIL\_HOST='smtp.gmail.com'

EMAIL\_PORT=25

EMAIL\_HOST\_USER='colocarcorreodegmail@gmail.com'

EMAIL\_HOST\_PASSWORD='colocarcontrasenadelcorreo'

EMAIL\_BACKEND='django.core.mail.backends.smtp.EmailBackend'

26) Regresamos al archivo urls del proyecto para agregar las rutas para el cambio de contraseña, si ejecutamos el proyecto tal cual como está, nos arrojará un error, precisamente porque necesitamos colocar las rutas

  # las rutas para el reseteo de las contraseñas

    path('reset/password\_reset/', PasswordResetView.as\_view(template\_name='registration/password\_reset\_form.html',

        email\_template\_name='registration/password\_reset\_email.html'),

        name='password\_reset'),

    path('reset/password\_resetDone/', PasswordResetDoneView.as\_view(template\_name='registration/password\_reset\_done.html'),

        name='password\_reset\_done'),

    path('reset/(?P<uidb64>[0-9A-Za-z\_\-]+)/(?P<token>.+)/', PasswordResetConfirmView.as\_view(template\_name='registration/password\_reset\_confirm.html'),

        name='password\_reset\_confirm'),

    path('reset/password\_resetComplete/', PasswordResetCompleteView.as\_view(template\_name='registration/password\_reset\_complete.html'),

        name='password\_reset\_complete'),

podemos probar antes si nos permite registrar un nuevo usuario, usamos el manage.py runserver y nos vamos a la ruta para registrar un nuevo usuario.

Nos vamos a base.html para cambiar una configuración del botón de cerrar sesión

 <li class="nav-item dropdown">

            <a class="nav-link dropdown-toggle" data-toggle="dropdown" href="#" id="themes">Opciones <span class="caret"></span></a>

            <div class="dropdown-menu" aria-labelledby="themes">

              <a class="dropdown-item" href="{% url 'logout' %}">Cerrar sesión</a>

            </div>

          </li>

Adicionamos la ruta para cerrar sesión

  path('logout/', logout\_then\_login, name='logout'),

**webgrafia**

1. <https://www.wearemarketing.com/es/blog/frameworks-en-el-desarrollo-web-las-mejores-practicas-para-tu-negocio-online.html>