Proyecto CRUD dptenis

Este proyecto se creó como proyecto de prueba para la limpieza de los datos capturados en una plantilla. Se utiliza el framework de Django para desarrollar la aplicación y como backend se utiliza el lenguaje de programación Python.

Lea el archivo README.md para detalles sobre la ejecución de proyecto de manera local.

Desarrollo de la aplicación:

La aplicación se creó dentro de una carpeta principal llamada “dptenis”, dentro de ella se ejecutó el siguiente comando de django para crear las carpetas y archivos base del proyecto

django-admin startproject dpTenis

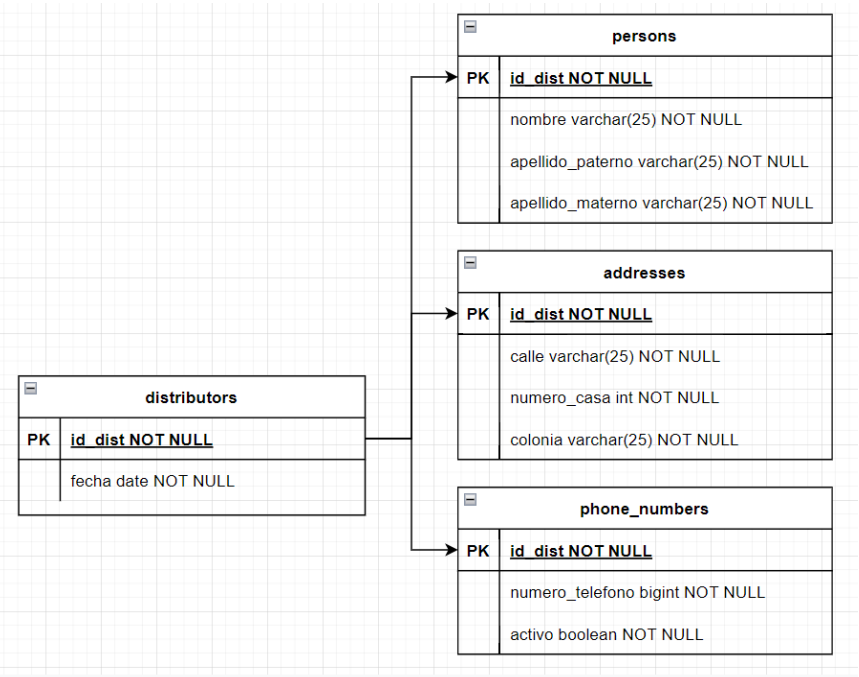
En la carpeta principal (dptenis) se crea la carpeta “apps” que contiene los archivos y carpeta base de la aplicación; se ejecuta el siguiente comando dentro de la carpeta “apps” para crear los archivos:

python manage.py startapp dptenisapp

Se aplicaron algunas configuraciones adicionales en el archivo settings.py:

En la sección LANGUAGE_CODE cambiar a es-mx
En la sección TIME_ZONE cambiar a America/Mazatlan

Base de datos dptenis



Tablas:

distributors: Tabla que contiene los id de los distribuidores y la fecha de registro.

- id_dist: Id del distribuidor.
- fecha: Fecha de alta o modificación del registro.

persons: Tabla que contiene la información del nombre completo de los distribuidores.

- id_dist: Id del distribuidor.
- nombre: Nombre o nombres.
- apellido_paterno: Apellido paterno.
- apellido_materno: Apellido materno.

addresses: Tabla que contiene la información del domicilio de los distribuidores.

- id_dist: Id del distribuidor.
- calle: Nombre de la calle del domicilio.
- numero_casa: Número de la casa.
- colonia: Nombre de la colonia del domicilio.

phone_numbers: Tabla que contiene la información de los número de teléfono de los distribuidores.

- id_dist: Id del distribuidor.
- numero_telefono: Número de telefono.
- activo: indica si el número está activo o no.

SQLite3

El motor de base de datos que se ocupa para el proyecto es SQLite, el cual viene como motor de base de datos por default de Django.

Django utiliza modelos, los cuales se utilizan como puente para crear, editar y consultar las tablas de la base de datos.

En el archivo models.py de la carpeta apps\dptenisapp, se agrega el siguiente código para crear los modelos:

```
class distributor(models.Model):
    id_dist = models.CharField(primary_key=True, max_length=25)
    fecha = models.DateTimeField()

class persons(models.Model):
    id_dist = models.CharField(max_length=25)
    nombre = models.CharField(max_length=25)
    apellido_paterno = models.CharField(max_length=25)
    apellido_materno = models.CharField(max_length=25)

class addresses(models.Model):
```

```
    id_dist = models.CharField(max_length=25)
    calle = models.CharField(max_length=25)
    numero_casa = models.PositiveIntegerField()
    colonia = models.CharField(max_length=25)

class phone_numbers(models.Model):
    id_dist = models.CharField(max_length=25)
    numero_telefono = models.PositiveBigIntegerField()
    activo = models.BooleanField()
```

En el archivo admin.py de la carpeta apps\dptenisapp se agrega el siguiente código para dar acceso a los modelos creados en el punto anterior.

```
from .models import *

admin.site.register(distributor)
admin.site.register(persons)
admin.site.register(addresses)
admin.site.register(phone_numbers)
```

Django tomará las configuraciones para crear las tablas en la base de datos.

Los siguientes comandos se utilizan para migrar las clases como tablas en la base de datos de MongoDB:
python manage.py migrate
python manage.py makemigrations

Se debe crear un superusuario para la administración de la app con el siguiente comando:

python manage.py createsuperuser

Username: dptenis
email: <vacio>
password: admin

Se ejecuta de nuevo la migración para terminar de configurar la aplicación:
python manage.py migrate

Para correr la aplicación de manera local, ejecutar el siguiente comando:

python manage.py runserver

Escribir en el navegador de internet http://127.0.0.1:8000 para ver la página http://127.0.0.1:8000/admin para entrar a la pantalla de administración.

Formulario y consulta con templates html y Jinja

Para capturar la información y alimentar las tablas de la base de datos, utilizamos formularios HTML, apoyándonos con Jinja para mostrar la información en la plantilla de forma más dinámica.

Dentro de la carpeta dptenisapp, creamos la carpeta templates, se coloca este nombre para evitar configuraciones adicionales ya que es el nombre que Django reconoce por default para los templates. Dentro de ella se crean los archivos html que contienen los templates.

Dentro de la carpeta dptenisapp, se crea un archivo llamado urls.py para gestionar las Url's de la aplicación agregando el siguiente código:

```
from django.urls import path

from . import views

urlpatterns = [
    path('', views.home),
    path('registradist/', views.registraDist),
    path('editardist/<_id_dist>', views.editarDist),
    path('editdist/', views.editDist),
    path('borradist/<_id_dist>', views.borraDist),
]
```

views.py - CRUD y limpieza de datos

La esencia del proyecto radica en el archivo views.py, el cual contiene las instrucciones necesarias para el CRUD y la limpieza de los datos.

Funciones

limpiaTexto: esta función es la encargada de limpiar el texto al momento de consultar la información en las tablas, eliminando caracteres especiales, números y vocales duplicadas en el texto:

```
def limpiaTexto(texto):

    # Se crea un diccionaro don se mapea los numeros correspondientes a cada letra de abecedario similar a ese número
    dic_num2let = {"0" : "o" , "1": "i", "3": "e", "4": "a", "5": "s", "6": "g", "7": "t"}
```

```
  # Lista de vocales para evitar vocales repetidas juntas
  list_vocals = {'a', 'e', 'i', 'o', 'u'}

  # Lista para validar que el texto solo contenga letras
  list_validate = re.compile("[A-Za-z]")

  txt = texto.split(' ') #se separa el texto por palabras para realizar el limpiado por cada una

  res = [] # declaración de una variable tipo lista donde se guardará el resultado de la limpieza

  # ciclo para recorrer cada palabra en la lista txt
  for el in txt:
    list_txt = [] #declaración de una variable tipo lista donde se guardará cada una de las letras que conforma la palabra ya limpia
    a = '' #declaración de una variable para guardar la letra anterior y validad que no sea un duplicado

    # ciclo para recorrer cada letra de la palabra
    for idx, l in enumerate(el):
      try: #buscamos la letra en el diccionario dic_num2let y en caso de encontrarla, guarda la letra correspondiente en una variable
        x = dic_num2let[l]
      except: #en caso de que el elemento buscado no se encuentre en el diccionario, es decir, que no sea un número, se guarda la letra en minúsculas
        x = l.lower()

      if idx == 0: #se valida si el el elemento es la primera letra de la palabra para convertirla en mayúscula
        x = x.upper()

      if list_validate.fullmatch(x): #se valida que el elemento sea una letra
        if x.lower() != a.lower() or (x.lower() not in list_vocals): #se valida que el elemento actual (x) no sea igual al elemento anterior (a), además, se valida que no sea una vocal.
          list_txt.append(x) #agregamos la letra a la lista
          a = x #se guarda el elemento actual (x) en una variable (a) para compararle en la siguiente iteración

    res.append(''.join(list_txt)) #agregamos la palabra ya limpia en la lista para formar la palabra completa
```

```
  return (' '.join(res)) #retornamos la palabra completa separada por espacios para mostrarla en el template
```

home: Esta función, se utiliza para mandar la información al template principal, y mostrar los registros en pantalla:

```
def home(request):

  # se obtiene la información de todos los modelos(tablas)
  _distributor=distributor.objects.all()
  _persons=persons.objects.all()
  _addresses=addresses.objects.all()
  _phone_numbers=phone_numbers.objects.all()

    _distrOptimize = [] #variable tipo lista para guardar la información a la cual se le aplicó la limpieza de datos
    _distComplete = [] #varia tipo lista para guardar la información real que se encuentra en las tablas

     #ciclo para iterar sobre cada uno de los registros de la tabla distributor para obtener la información para cada Id
  for d in _distributor:

    # se busca el id de distribuido de cada una de las tablas para obtener su información
    _persons=persons.objects.get(id_dist=d.id_dist)
    _addresses=addresses.objects.get(id_dist=d.id_dist)
    _phone_numbers=phone_numbers.objects.get(id_dist=d.id_dist)

        #se guarda la información en la lista optimizada, aplicando limpieza a cada uno de los campos de texto con la función limpiaTexto()
    _distrOptimize.append(
      {
        "id_dist" : d.id_dist,
                                  "nombre_completo":  limpiaTexto(_persons.nombre) + " " + limpiaTexto(_persons.apellido_paterno) + " " + limpiaTexto(_persons.apellido_materno),
                               "direccion"  :  "Calle:  "  +  limpiaTexto(_addresses.calle)  +  ",  #"  +  str(_addresses.numero_casa) + ", Colonia: " + limpiaTexto(_addresses.colonia),
        "telefono" : _phone_numbers.numero_telefono
```

```
        }
      )

        #se guarda la información real de cada tabla para regresar un listado único que pueda ser facimelte consumido por el template.
      _distComplete.append(
        {
          "id_dist" : d.id_dist,
          "nombre": _persons.nombre,
          "apellido_paterno" : _persons.apellido_paterno,
          "apellido_materno" : _persons.apellido_materno,
          "calle" : _addresses.calle,
          "numero_casa" : _addresses.numero_casa,
          "colonia" : _addresses.colonia,
          "numero_telefono" : _phone_numbers.numero_telefono
        }
      )

   #retornamos el renderizado del template principal, y le mandamos las listas creadas
       return  render(request,  "dpteniscrud.html",  {"distributor"  :  _distrOptimize,  "distComplete"  : _distComplete})
```

registraDist: Esta función se utiliza para guardar un registro nuevo en la tabla

```
def registraDist(request):

   list_validate = re.compile("[A-Za-z0-9]+")

   #validamos si el id existe en la tabla distributors
     if  not(list_validate.fullmatch(request.POST['txtIdDist'])): #en caso de que exista regresa un mensaje al front por la duplicidad
     messages.success(request, 'El id del distribuidor solo debe contener números y letras')
     return redirect('/')

    #optenemos la información de los inputs del formulario y los guardamos en variables que hacen referencia a cada campo de las tablas
   _id_dist=request.POST['txtIdDist']
   _nombre=request.POST['txtNombre']
   _apellido_paterno=request.POST['txtApPaterno']
   _apellido_materno=request.POST['txtApMaterno']
   _calle=request.POST['txtCalle']
```

```
  _numero_casa=request.POST['txtNumeroCasa']
  _colonia=request.POST['txtColonia']
  _numero_telefono=request.POST['txtTelefono']

  try: #se guarda la información en las tablas
    _distributor = distributor.objects.create(
      id_dist=_id_dist,
      fecha = datetime.datetime.now()
    )
    _persons = persons.objects.create(
      id_dist=_id_dist,
      nombre=_nombre,
      apellido_paterno=_apellido_paterno,
      apellido_materno=_apellido_materno
    )
    _addresses = addresses.objects.create(
      id_dist=_id_dist,
      calle=_calle,
      numero_casa=_numero_casa,
      colonia=_colonia
    )
    _phone_numbers = phone_numbers.objects.create(
      id_dist=_id_dist,
      numero_telefono=_numero_telefono,
      activo=True
    )

    messages.success(request, '¡Distribuidor registrado!')

    return redirect('/')
  except: #en caso de el proceso de guardado arroje un error, manda un mensaje al front
    messages.success(request, '¡El id del distribuidor ya se encuentra registrado!')

    return redirect('/')
```

editarDist: Esta función, se utiliza para mostrar la información que se desea modificar:

```
def editarDist(request, _id_dist):

  #optenemos la información de cada tabla buscada por medio del id
  _distributor=distributor.objects.get(id_dist=_id_dist)
```

```
    _persons=persons.objects.get(id_dist=_id_dist)
    _addresses=addresses.objects.get(id_dist=_id_dist)
    _phone_numbers=phone_numbers.objects.get(id_dist=_id_dist)

    #guardamos la información en una variable tipo diccionario para ser consumida facilmente por el template de edición
    _distrComplete = {
        "id_dist" : _distributor.id_dist,
        "nombre": _persons.nombre,
        "apellido_paterno" : _persons.apellido_paterno,
        "apellido_materno" : _persons.apellido_materno,
        "calle" : _addresses.calle,
        "numero_casa" : _addresses.numero_casa,
        "colonia" : _addresses.colonia,
        "numero_telefono" : _phone_numbers.numero_telefono
    }

    return render(request, "editardist.html", {"distributor": _distrComplete})
```

editDist: Función que sirve para guardar la información del registro modificado en el template de edición:

```
def editDist(request):

    #optenemos la información de los inputs del formulario y los guardamos en variables que hacen referencia a cada campo de las tablas
    _id_dist=request.POST['txtIdDist']
    _nombre=request.POST['txtNombre']
    _apellido_paterno=request.POST['txtApPaterno']
    _apellido_materno=request.POST['txtApMaterno']
    _calle=request.POST['txtCalle']
    _numero_casa=request.POST['txtNumeroCasa']
    _colonia=request.POST['txtColonia']
    _numero_telefono=request.POST['txtTelefono']

    #optenemos la información a editar de cada tabla buscada por medio del id
    _distributor=distributor.objects.get(id_dist=_id_dist)
    _persons=persons.objects.get(id_dist=_id_dist)
    _addresses=addresses.objects.get(id_dist=_id_dist)
    _phone_numbers=phone_numbers.objects.get(id_dist=_id_dist)
```

```
    #se actualiza la información de cada campo en la tabla correspondiente
    _distributor.fecha = datetime.datetime.now()

    _persons.nombre = _nombre
    _persons.apellido_paterno = _apellido_paterno
    _persons.apellido_materno = _apellido_materno

    _addresses.calle = _calle
    _addresses.numero_casa = _numero_casa
    _addresses.colonia = _colonia

    _phone_numbers.numero_telefono = _numero_telefono

    #guarda los cambios en cada tabla
    _distributor.save()
    _persons.save()
    _addresses.save()
    _phone_numbers.save()

    messages.success(request, '¡Registro Modificado!')

    return redirect('/')
```

borraDist: Función que se utiliza para borrar un registro de las tablas:

```
def borraDist(request, _id_dist):
    #obtenemos el registro de cada tabla por medio del Id y aplicamos el borrado de ese registro
    _distributor=distributor.objects.get(id_dist=_id_dist)
    _distributor.delete()
    _persons=persons.objects.get(id_dist=_id_dist)
    _persons.delete()
    _addresses=addresses.objects.get(id_dist=_id_dist)
    _addresses.delete()
    _phone_numbers=phone_numbers.objects.get(id_dist=_id_dist)
    _phone_numbers.delete()

    messages.success(request, '¡Registro Eliminado!')

    return redirect('/')
```

Uso

Se capturaron 5 registros con errores en nombre, apellidos, nombre de calles y nombre de colonias para probar la limpieza de los datos con la función “limpiaTexto()” obteniendo los siguientes resultados:

Lista original:

#	Id Distribuidor	Nombre	Apellido Paterno	Apellido Materno	Calle	Número de Casa	Colonia	Télefono
1	asd123456789	d4v1id	4r4ngur3	m00or4l3s	Tul1p4n35	1715	1234567890	1234567890
2	safasf251358	j3sUs d4V1d	AraAangur3	M0ora4l3e5s	Ju570 s13Rraa	852	4859632784	4859632784
3	as1258746985	alexy	v1rg3n	vizc4rr4	tulipanes	1715	2578964235	2578964235
4	new125874693	m1guel 4ng3el	ar4Nguur.-e	M0Or4l3es	Clivi1a4-.-.,	4721	1234567890	1234567890
5	rty212523684	d4vid 4l3x4nd.-3r	Ar4angur3	v1rgen	tulipanes	1715	1258745693	1258745693

En la consulta original, se observan números y caracteres no deseados en los campos nombre, apellido paterno, apellido materno, calle y colonia. Se aplicó la limpieza de estos campos y obteniendo el siguiente resultado:

#	Id Distribuidor	Nombre	Dirección	Télefono
1	asd123456789	David Arangure Morales	Calle: Tulipanes, #1715, Colonia: Colinas Del Bosque	1234567890
2	safasf251358	Jesus David Arangure Moraless	Calle: Justo Sierra, #852, Colonia: Lopez Mateos	4859632784
3	as1258746985	Alexy Virgen Vizcarra	Calle: Tulipanes, #1715, Colonia: Colinas Del Bosque	2578964235
4	new125874693	Miguel Angel Arangure Morales	Calle: Clivia, #4721, Colonia: Floress Magon	1234567890
5	rty212523684	David Alexander Arangure Virgen	Calle: Tulipanes, #1715, Colonia: Colinas Del Bosque	1258745693

En la consulta optimizada, se puede observar cómo se eliminan los caracteres especiales, letras duplicadas (solo vocales), y la sustitución de números por letras, además, se muestra la información en una columna con el nombre completo (Nombre) y el domicilio completo (Domicilio).

Conclusiones

Si bien se aplica un limpieza básica en los campos de texto para eliminar los caracteres especiales o no deseados, se necesita un diccionario de datos más complejo para comparar nombres de colonias, calles y verificar si las consonantes repetidas de manera consecutiva, sean correctas en la palabra, tal como la doble “L” y la doble “R”.