Correcta Organización de Carpetas

Dentro de la carpeta requirements voy a alojar todos los requerimientos que va a necesitar mi proyecto para ejecutarse

- Para ver los paquetes instalados en mi entorno local uso pip freeze --local
- Para instalar los paquetes desde el txt uso: pip install -r .\local.txt

Archivo Secret, para esconder información sensible en el proyecto

A fín de poder esconder información sensible tales como los entornos donde esta la conexion a la BD pues se usa un archivo JSON secret a la altura del manager

En este archivo estara la info sensible a la cual yo llamaré desde cualquier archivo siempre que sea necesario y al ser un diccionario pues le llamaria por su clave respectivamente

Secret.json y Gitignore

Se crea un Gitignore donde especifico los archivos que no quiero que se suban al repositorio por seguridad

Aplicaciones y AbstractBaseUser

Siempre que se trabaje con usuarios el nombre de aplicacion se recomienda que sea users

Para el caso de usuarios Django nos facilita una forma sencilla para crear los modelos haciendo uso de la clase interna AbstractUser

Sumado a esto para poder tener un control mas robusto de todos los usuarios de Django lo que se utiliza es una clase llamada PermissionsMixin

La ventaja de usar esta clase es que le indicamos a Django que unicamente trabaje con el modelo creado por nostros y no con el interno, además de poder hacer un seguimiento de todos los superusuarios

Se debe configurar en el base.py indicando que ahora voy a trabajar con otro modelo

Además al redefinir el modelo Usuario debo indicarle a Django cuales son las variables o atributos de autenticacion.
Para ello lo especifico en el modelo que atributo hace referencia a USERNAME_FIELD

Sin embargo tambien se requiere de managers para que funcione correctamente lo que se hará en la siguiente clase

Escribiendo User Managers

En mi archivo managers.py importo el BaseUserManager

A fin de poder indicar si el usuario que se crea es superusuario y si puede acceder al admin lo que se haria es redefinir el modelo

Pero si he puesto en mi modelo otros atributos como email pues los debo indicar para que se cree el objeto, y eso lo hago con el
REQUIRED_FIELDS = ['ATRIBUTO','ATRIBUTO']

Para el atrbuto is_staff se debe redefinir el modelo

Crear Usuario - FormView

Para el caso de la creacion de usuarios se usará el formview y nos apoyaremos de las funciones realizadas en el manager

Como tarea se mando validar la longitud del campo si bien si se lo realizó por alguna razón al activar esta validacion se desactiva la otra de que las contraseñas coincidan.
Sin embargo existe otra forma de validar

Login User

Se creo un formulario normal en los forms , y en las vistas se lo llamó.
Haciendo uso del authenticate y del login pues se hace la comparativa entre los datos mandados para hacer el login

Cerrar Sesión - Logout

Al ser un proceso simple pues requiere de una vista muy sencilla , para ello se llamará al View padre de todos que internamente no tiene nada más que una simple vista, una simple función que se ejecuta, recibe algo y ejecuta algo, nada más.

Al ser muy simple el View no necesita de un template, pero si se puede sobreescribir funciones básicas como el GET y el POST

Sumado a ésto también del paquete django.contrib.auth importo el logout

Una vez cerrado sesion se debe redireccionar a otra página, para ello se hace uso del paquete
from django.http import HttpResponseRedirect
y en la vista creada al retornar se manda en los parametros el reverse:
función que nos facilita el poder navegar entre las urls de nuestro sistema

Diferencia entre reverse y reverse_lazy :

Por su traducción perezoso

el reverse_lazy y el reverse es lo mismo solo que el lazy espera a que cargen todos los archivos o leer todos los urls.py para una vez hecho ésto realizar el respectivo redireccionamiento.
Como python es dinámico es posible que en el momento que se ejecuta el urls.py aún no se haya cargado (aunque se lo hará más adelante) pero ésto arrojará un error.
Es aquí cuando reverse_lazy puede ser útil

Validación en Formulario Login

Se realizará una validación directamente en el form del usuario si es o no un usuario de tal forma que ya ni siquiera entre a las vistas y ésto haga un rendimiento mucho más rápido

para acceder al usuario de la sesion en el template se usa {{user.username}}

Mixins Teoria

Un Mixin se resumen en herencia de vistas, sin embargo a diferencia de la herencia normal tiene una pequeña particularidad que cuando nosotros teníamos nuestro modelo,
la particularidad es que al momento de hacer la herencia especifico varios mixins
vista(mix1, mix2, formView)

Esto es necesario puesto que una vista o varias vistas va a necesitar de varios mixins para ahorrarse líneas de código

Estos Mixins los construimos nosotros o algunos ya los trae Django como el Django login required mix

Mixins Ejemplo Práctico

Se hará en la app home
Contexto: Necesitamos enviar una fecha en todas nuestras vistas, entonces para no repetir mismo código haremos uso del Mixin

Para declarar se manda como parametro (object)

Login Required Mixin

El requerimiento es que era que cuando entremos a una página solo se la pueda acceder si se está logueado

Django ya nos ofrece un login mixin para este tipo de situaciones

Al importar el LoginRequiredMixin se debe poner un atributo o propiedad la cual indica que va a suceder cuando se intente ingresar a la vista pero no estan logeados, por lo tanto nos pide a donde nos va a enviar si no está logeado , y éste atributo se llama login_url

Actualizando Password de Usuario

Se recomienda siempre que se actualicen datos pues hacerlo por separado lo que es contraseña es decir por vistas separadas, una vista aparte para lo que es actualizar contraseña y una vista aparte para lo que es modificar datos del usuario

1) Verificar que al ingresar la contraseña actual coincida con el usuario activo
Para ello se obtiene al usuario de la sesion con self.request.user
2)