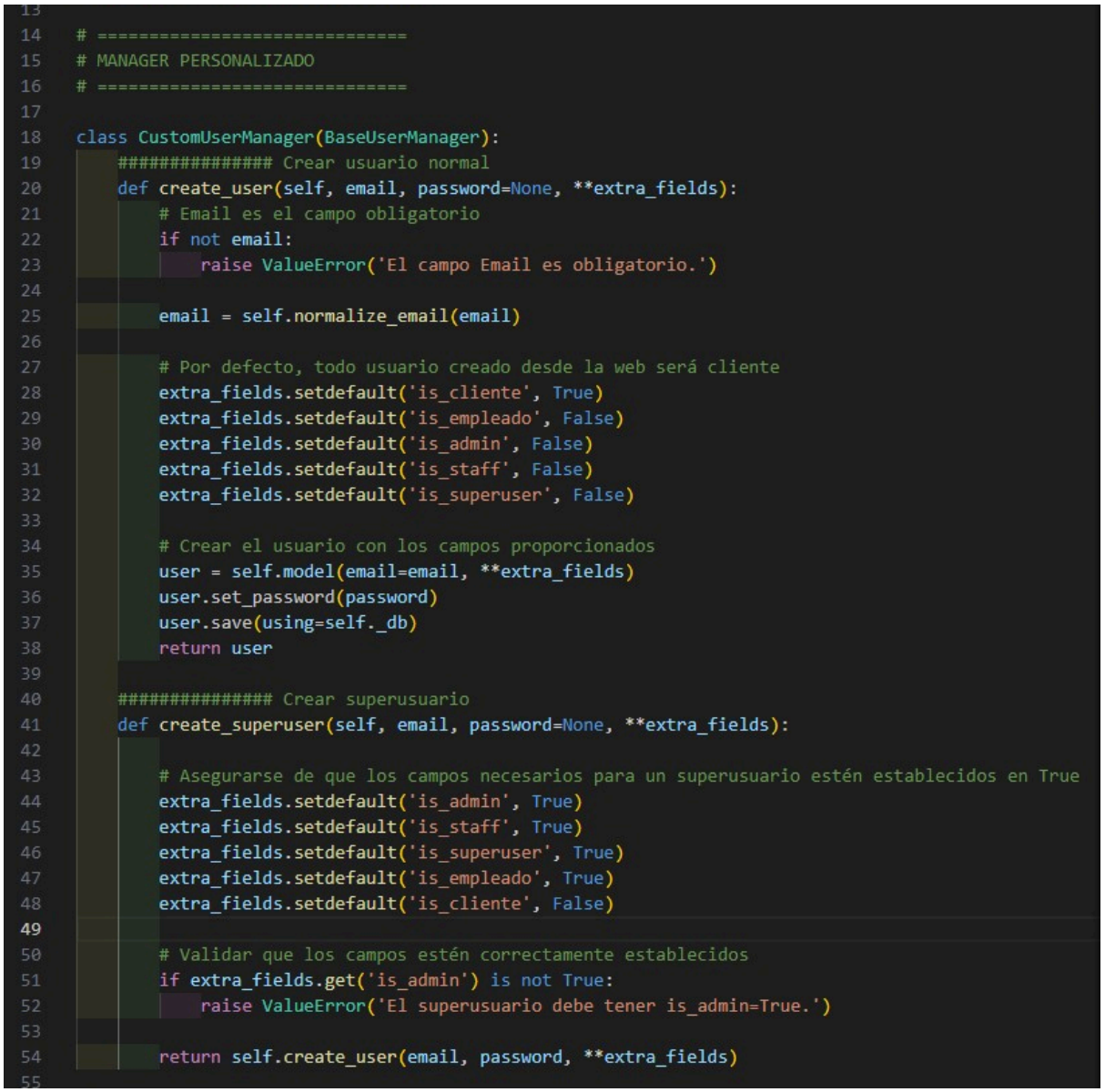
Explicación de los Modelos Django del Proyecto: Peluqueria Martita

.Class CustomUserManager(BaseUserManager):

Este modelo “CustomUserManager” (con “BaseUserManager” es una clase que Django trae por defecto para la personalización de un manager para controlar la creacion de usuarios) es un Administrador de Usuarios que trae Django por defecto en su instalación , será modificado para poder hacer una diferencia entre nuestros Modelos de Usuario en donde se agregaran nuestros otros Modelos Empleado y Cliente, Así con esta Propia Personalización o Modificación de este Modelos crearemos una diferencia y controlamos cómo los **“Usuarios Normales”(create_user)** y los **“Super Usuarios”(create_superuser)** se Registren/Logueen a la Base de Datos.



```python
# ==============================
# MANAGER PERSONALIZADO
# ==============================

class CustomUserManager(BaseUserManager):
    ############### Crear usuario normal
    def create_user(self, email, password=None, **extra_fields):
        # Email es el campo obligatorio
        if not email:
            raise ValueError('El campo Email es obligatorio.')

        email = self.normalize_email(email)

        # Por defecto, todo usuario creado desde la web será cliente
        extra_fields.setdefault('is_cliente', True)
        extra_fields.setdefault('is_empleado', False)
        extra_fields.setdefault('is_admin', False)
        extra_fields.setdefault('is_staff', False)
        extra_fields.setdefault('is_superuser', False)

        # Crear el usuario con los campos proporcionados
        user = self.model(email=email, **extra_fields)
        user.set_password(password)
        user.save(using=self._db)
        return user

    ############### Crear superusuario
    def create_superuser(self, email, password=None, **extra_fields):

        # Asegurarse de que los campos necesarios para un superusuario estén establecidos en True
        extra_fields.setdefault('is_admin', True)
        extra_fields.setdefault('is_staff', True)
        extra_fields.setdefault('is_superuser', True)
        extra_fields.setdefault('is_empleado', True)
        extra_fields.setdefault('is_cliente', False)

        # Validar que los campos estén correctamente establecidos
        if extra_fields.get('is_admin') is not True:
            raise ValueError('El superusuario debe tener is_admin=True.')

        return self.create_user(email, password, **extra_fields)
```

1. **Creación de un Usuario Normal:**

def create_user(self, email, password=None, **extra_fields):

```python
############### Crear usuario normal
def create_user(self, email, password=None, **extra_fields):
    # Email es el campo obligatorio
    if not email:
        raise ValueError('El campo Email es obligatorio.')

    email = self.normalize_email(email)

    # Por defecto, todo usuario creado desde la web será cliente
    extra_fields.setdefault('is_cliente', True)
    extra_fields.setdefault('is_empleado', False)
    extra_fields.setdefault('is_admin', False)
    extra_fields.setdefault('is_staff', False)
    extra_fields.setdefault('is_superuser', False)

    # Crear el usuario con los campos proporcionados
    user = self.model(email=email, **extra_fields)
    user.set_password(password)
    user.save(using=self._db)
    return user
```

- **Validación de Email como campo obligatorio, para evitar crear Usuarios sin Email:** Tomará el Email como un campo obligatorio a la cual Completar, normalize_email solo será para convertir el email en un formato normal estandarizado (no es importante saber esto, pero evita duplicidad de emails si se escriben en mayuscula o minuscula)

```python
if not email:
    raise ValueError('El campo Email es obligatorio.')

email = self.normalize_email(email)
```

- **Campos/Casillas que definiran los Roles (y se mostrarán en el admin):** Django trae en su models.py estas casillas por defecto, nosotros la modificamos para que cada Usuario Normal se registre desde el formulario Login.html de nuestra Web por Defecto este será un “Usuario Cliente” (extra_fields.setdefault(‘is_cliente’, True)), por eso el Único Marcado como True será is_cliente mientras que los demás como empleado,admin y superuser serán False

```python
# Por defecto, todo usuario creado desde la web será cliente
extra_fields.setdefault('is_cliente', True)
extra_fields.setdefault('is_empleado', False)
extra_fields.setdefault('is_admin', False)
extra_fields.setdefault('is_staff', False)
extra_fields.setdefault('is_superuser', False)
```

- **Guardado del Usuario Registrado:**

```python
# Crear el usuario con los campos proporcionados
user = self.model(email=email, **extra_fields)
user.set_password(password)
user.save(using=self._db)
return user
```

2. Creación de un Usuario SuperUsuario:

def create_superuser(self, email, password=None, **extra_fields):

```python
############### Crear superusuario
def create_superuser(self, email, password=None, **extra_fields):

    # Asegurarse de que los campos necesarios para un superusuario estén establecidos en True
    extra_fields.setdefault('is_admin', True)
    extra_fields.setdefault('is_staff', True)
    extra_fields.setdefault('is_superuser', True)
    extra_fields.setdefault('is_empleado', True)
    extra_fields.setdefault('is_cliente', False)

    # Validar que los campos estén correctamente establecidos
    if extra_fields.get('is_admin') is not True:
        raise ValueError('El superusuario debe tener is_admin=True.')

    return self.create_user(email, password, **extra_fields)
```

- **Campos/Casillas que definiran los Roles (y se mostrarán en el admin):**
  Esta vez Todas las casillas serán **“True”** a excepción del is_cliente que será **“False”** haciendo que cada Usuario que se Registre con el Comando de Ejecucion de Django **“py manage.py createsuperuser”** sea tanto empleado,admin y superusuario
  Haciendo así una diferencia entre el registro de la página web y el registro en el superuser

```python
# Asegurarse de que los campos necesarios para un superusuario estén establecidos en True
extra_fields.setdefault('is_admin', True)
extra_fields.setdefault('is_staff', True)
extra_fields.setdefault('is_superuser', True)
extra_fields.setdefault('is_empleado', True)
extra_fields.setdefault('is_cliente', False)
```

- **is_admin forzado a que para un superusuario sea “True” (sin necesidad):**
  evita que el superusuario se cree como un **“False”** para el admin, si se falla no creara el superusuario

```python
# Validar que los campos estén correctamente establecidos
if extra_fields.get('is_admin') is not True:
    raise ValueError('El superusuario debe tener is_admin=True.')
```

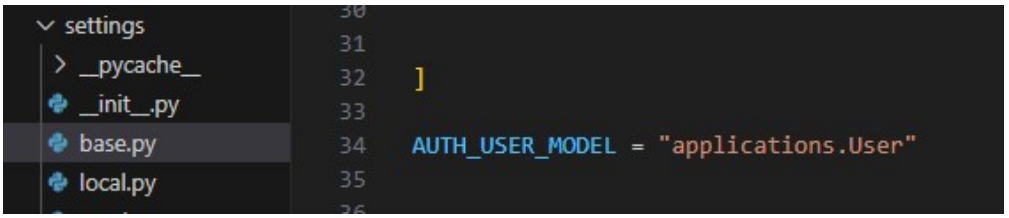
- **Guardado del Usuario Registrado:**

```python
return self.create_user(email, password, **extra_fields)
```

Return Devolverá o volverá a llamar al guardado de Usuario que se utiliza en create_user, para guardar el superuser sin la necesidad de volver a construir de nuevo el código como algo repetitivo

```python
# Crear el usuario con los campos proporcionados
user = self.model(email=email, **extra_fields)
user.set_password(password)
user.save(using=self._db)
return user
```

.Class User(AbstractUser):



Django por defecto en su Base de Datos trae un Abstract User, al modificar/Crear nuestra propia Clase User(Abstractuser) estamos personalizando “solo un poco” este modelo de usuario para el registro de Usuarios(y que pronto lo implementaremos a nuestros Modelos Empleado y Cliente)
Además que anulamos algunas cosas y las reemplazamos por otras para darles prioridad (como se verá en el siguiente punto)

```python
# ============================
# MODELO DE USUARIO PERSONALIZADO
# ============================

class User(AbstractUser):

    # Eliminar los campos username, first_name y last_name de AbstractUser
    username = None
    first_name = None
    last_name = None

    # Definir email como el campo único de identificación
    email = models.EmailField('Correo electrónico', unique=True)
    nombre = models.CharField('Nombre', max_length=50)
    apellido = models.CharField('Apellido', max_length=50)

    # Campos para diferenciar roles
    is_cliente = models.BooleanField(default=False)
    is_empleado = models.BooleanField(default=False)
    is_admin = models.BooleanField(default=False)

    objects = CustomUserManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['nombre', 'apellido']

    def get_full_name(self):
        return f"{self.nombre} {self.apellido}".strip()

    def get_short_name(self):
        return self.nombre

    def __str__(self):
        rol = "Empleado" if self.is_empleado else (
              "Cliente" if self.is_cliente else "Usuario"
            )

        return f"{self.get_full_name()} ({rol}) - {self.email}"
```

- **Evitación y eliminación de duplicidad Nula en los campos que Django trae por defecto:**
  Anulamos estos campos y los cambiamos por otros campos(siguente punto) a los cuales le dimos prioridad, si no eliminamos estos campos por defecto en el admin de Django te

devolverá estos campos pero vacíos porque ya no tienen prioridad al cambiarlos por otros

```
# Eliminar los campos username, first_name y last_name de AbstractUser
username = None
first_name = None
last_name = None
```

- **Definición de email como campo principal de identificación de Usuario:**
  Se reemplazará username(que en el punto anterior se puso en None) campo de identificación por el Email con "unique=True"

```
# Definir email como el campo único de identificación
email = models.EmailField('Correo electrónico', unique=True)
```

- **Campos de nombre y apellido:**
  Se reemplazará first_name y last_name, esto servirá a futuro para no repetir casillas e implementarlos a los Modelos de Empleado y Cliente

```
nombre = models.CharField('Nombre', max_length=50)
apellido = models.CharField('Apellido', max_length=50)
```

- **Roles:**
  Se mantendrán en False noes necesario pero en CustomUserManager se redefinirán, esto sirve al agregar un usuario en admin

```
# Campos para diferenciar roles
is_cliente = models.BooleanField(default=False)
is_empleado = models.BooleanField(default=False)
is_admin = models.BooleanField(default=False)
```

- **Coneccion con El CustomUserManager:**

```
objects = CustomUserManager()
```

```
USERNAME_FIELD = 'email'
REQUIRED_FIELDS = ['nombre', 'apellido']
```

```python
def get_full_name(self):
    return f"{self.nombre} {self.apellido}".strip()

def get_short_name(self):
    return self.nombre

def __str__(self):
    rol = "Empleado" if self.is_empleado else (
        "Cliente" if self.is_cliente else "Usuario"
    )

    return f"{self.get_full_name()} ({rol}) - {self.email}"
```

.Conexion del User con nuestro [Forms.py](#) :

```python
class CustomUserCreationForm(UserCreationForm):
    class Meta:
        model = User
        fields = ['nombre', 'apellido', 'email', 'password1', 'password2']
        labels = {
            'nombre': 'Nombre',
            'apellido': 'Apellido',
            'email': 'Correo electrónico',
        }
```

para mas informacion de la modificación de modelos que se han realizado por favor vea los modelos que Django trae por defecto Dentro de su carpeta donde se instaló el “-m venv Entorno”:

“entorno\Lib\site-packages\django\contrib\auth\[models.py](#)”

podra ver todo el código que Django trae por defecto en su Instalación

.Nuestros Modelos Propios de nuestra Aplicación:

```python
class Cliente(models.Model):
    user = models.OneToOneField(
        User,
        on_delete=models.CASCADE,
        related_name='cliente',
        limit_choices_to={'is_cliente': True}, # solo usuarios con is_cliente=True
        )

    dni = models.CharField(max_length=8, verbose_name='DNI', null=True, blank=True)
    fecha_nacimiento = models.DateField('Fecha de nacimiento', default='2000-01-01')
    telefono = models.CharField(max_length=15, null=True, blank=True)
    domicilio = models.CharField('Domicilio', max_length=100, null=True, blank=True)

    def __str__(self):
        return f"{self.user.get_full_name()} - {self.user.email} "

    def edad(self):
        today = date.today()
        return (
            today.year - self.fecha_nacimiento.year
            - ((today.month, today.day) < (self.fecha_nacimiento.month, self.fecha_nacimiento.day))
        )
```

```python
class Servicios(models.Model):
    tipo_servicio = models.CharField('Tipo de servicio', max_length=50)
    costo = models.DecimalField('Costo', max_digits=8, decimal_places=2)
    observaciones = models.TextField('Observaciones', null=True, blank=True)

    def __str__(self):
        return f"{self.tipo_servicio} (${self.costo})"
```

```python
class Empleado(models.Model):
    user = models.OneToOneField(
        User,
        on_delete=models.CASCADE,
        related_name='empleado',
        limit_choices_to={'is_empleado': True}, # solo usuarios con is_empleado=True
        )
    dni = models.CharField(max_length=8, verbose_name='DNI', null=True, blank=True)
    fecha_nacimiento = models.DateField('Fecha de nacimiento', default='2000-01-01')
    telefono = models.CharField(max_length=15, null=True, blank=True)
    domicilio = models.CharField('Domicilio', max_length=100, null=True, blank=True)
    puesto = models.ForeignKey(Servicios, on_delete=models.SET_NULL, null=True, blank=True)

    def __str__(self):
        return f"{self.user.get_full_name()} - {self.puesto} - {self.user.email}"

    def edad(self):
        today = date.today()
        return (
            today.year - self.fecha_nacimiento.year
            - ((today.month, today.day) < (self.fecha_nacimiento.month, self.fecha_nacimiento.day))
        )
```

```python
class Turnos(models.Model):
    cliente = models.ForeignKey(Cliente, on_delete=models.CASCADE)
    empleado = models.ForeignKey(Empleado, on_delete=models.CASCADE)
    servicio = models.ForeignKey(Servicios, on_delete=models.CASCADE)
    fecha = models.DateField('Fecha', default='2000-01-01')
    hora = models.TimeField('Hora', default="09:00")
    observaciones = models.CharField(max_length=200, null=True, blank=True)

    def __str__(self):
        return (
            f"Turno del Cliente ''{self.cliente.user.get_full_name()}'' con "
            f" el Empleado ''{self.empleado.user.get_full_name()} ''"
            f" para el Servicio ''{self.servicio.tipo_servicio}'' el dia {self.fecha} a las {self.
            hora}"
        )
```

```python
# ======================================
# PARA CREAR PERFIL AUTOMÁTICO
# ======================================

@receiver(post_save, sender=User)
def crear_perfil_usuario(sender, instance, created, **kwargs):
    if created:
        if instance.is_cliente:
            Cliente.objects.create(user=instance)
        elif instance.is_empleado:
            Empleado.objects.create(user=instance)
```

También puede ver la base de datos de Django utilizando Visual Studio Code en “db.sqlite3”