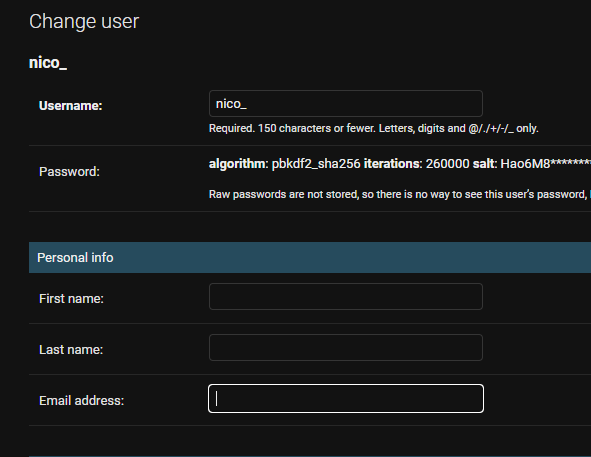
Playground Avanzado (Parte III)

**Edición de usuario**

El usuario no es otra cosa que una tabla de la base de datos que hace también a nuestro **model**, por lo cual se puede hacer CRUD, es más ya hicimos gran parte de **CRUD** sobre el usuario sin saberlo.

Principalmente, nos quedaría editar un usuario ya creado.

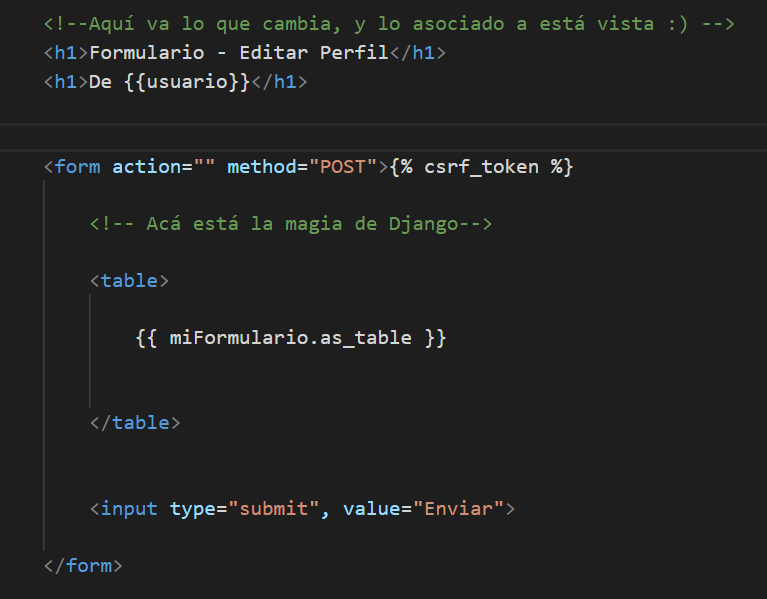
Recordemos que por defecto Django tiene los siguientes campos de usuario



Necesitaremos una **vista**, la **url** y el **template**, heredando de las funcionalidades de User que nos da Django.

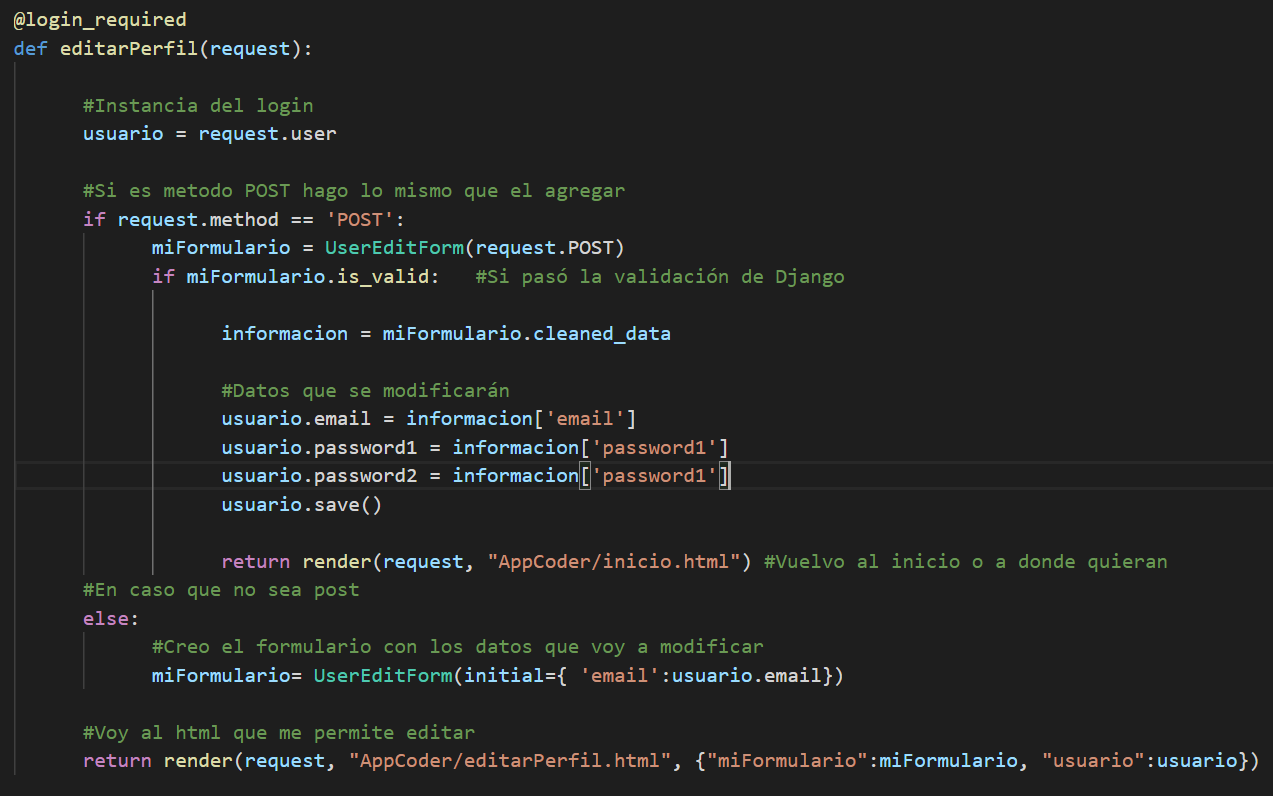
Primero **editarPerfil.html**

**Ver archivo: editarPerfil.txt**

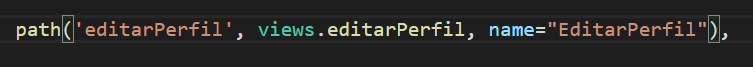


Nuestra vista

Ver archivo: editarPerfil\_views.txt

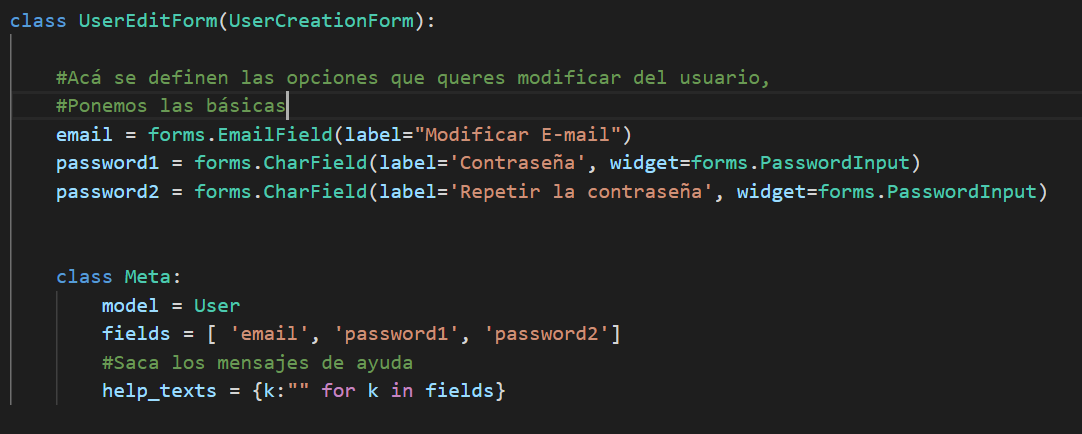


Urls



path('editarPerfil', views.editarPerfil, name="EditarPerfil"),

Form



Script:

# Clase 24, agregamos el UserEditForm

class UserEditForm(UserCreationForm):

# Obligatorios

email = forms.EmailField(label="Ingrese su email:")

password1 = forms.CharField(label='Contraseña', widget=forms.PasswordInput)

password2 = forms.CharField(

label='Repetir la contraseña', widget=forms.PasswordInput)

last\_name = forms.CharField()

first\_name = forms.CharField()

class Meta:

model = User

fields = ['email', 'password1', 'password2', 'last\_name', 'first\_name']

Veamos cómo modificar el registro de usuario para que además de sus datos básicos podamos agregar otros.

Solo modificando el forms.py



Veamos el impacto de las anteriores modificaciones en el templete y Django sin cambiar nada más.



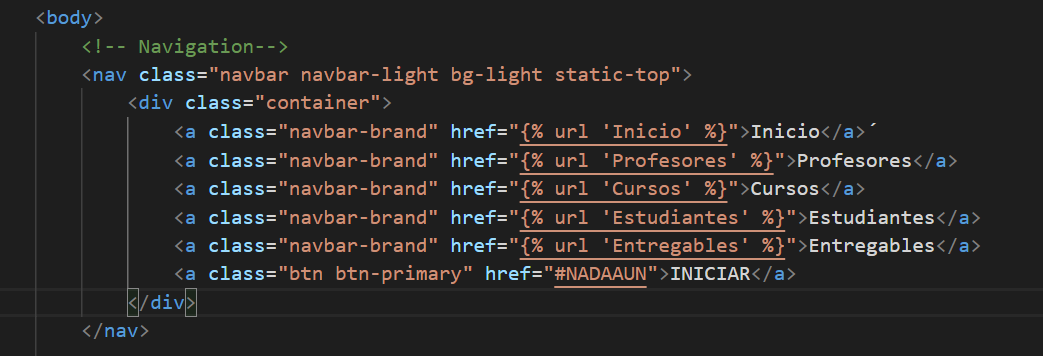
**Avatar**

Suele ser común en una página web que se pueda ver parte de la información del usuario registrado en todas las secciones de la web.

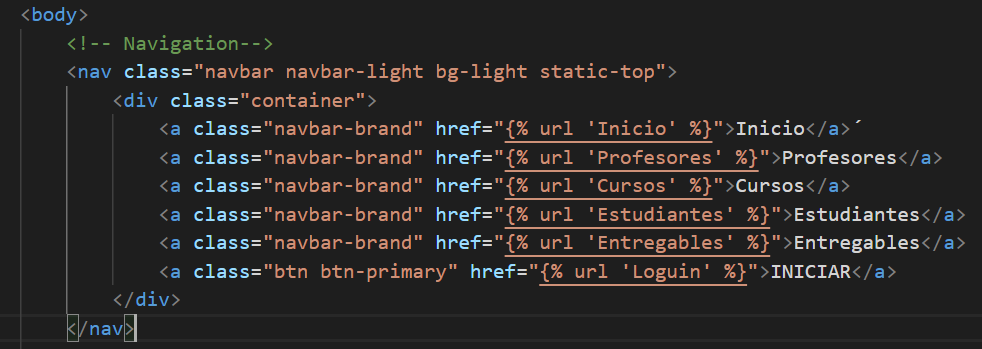
¿Cómo lo conseguimos?

Si queremos que la info esté siempre disponible lo debemos hacer en la plantilla Padre, la cual hereda a todas las otras.

Recordemos lo que teníamos.

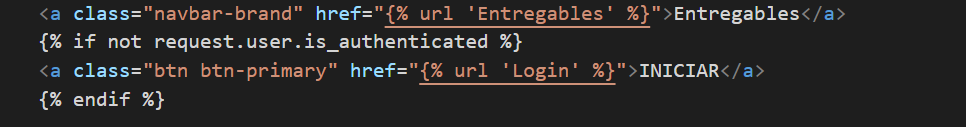


Habíamos dejado en suspenso el botón INICIAR para hacer login, así que ahora ya podemos referenciar a nuestro login 👇



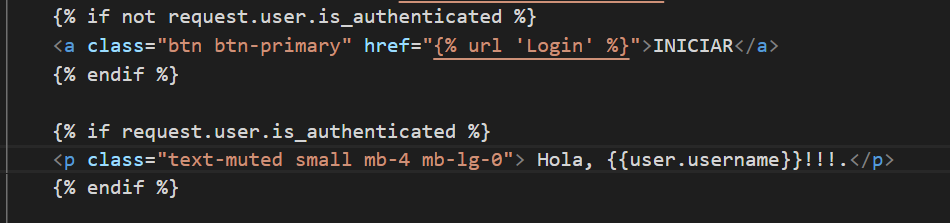
Pero no tiene sentido pedirle a alguien que haga Login si ya lo hizo

👇



Mostramos el Login solo en el caso que la persona no está logueada.

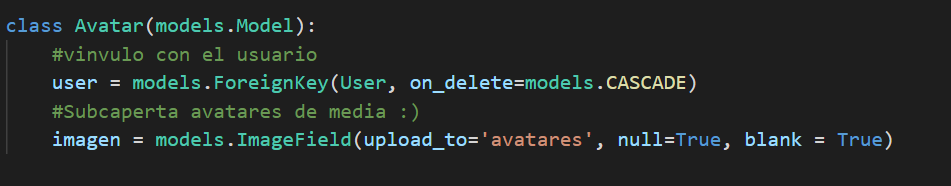
Pero si ya está logueada podemos mostrar su info.



Otra cosa interesante que se suele hacer con un Usuario es asignarle una foto, y eso es justamente lo que define al Avatar en una web.

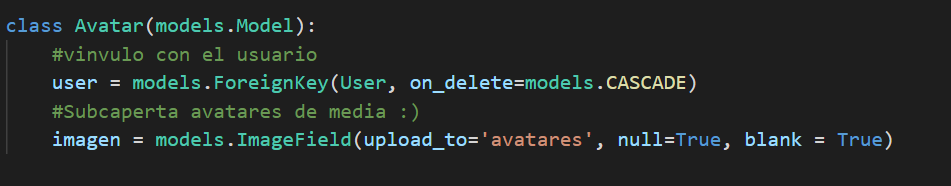
Así que veamos cómo hacer eso

👇



Se genera un nuevo tipo de dato ImageField y se lo guarda en la carpeta “avatares”.

Se generó un vínculo entre la clase Avatar y la clase User



Script:

from django.contrib.auth.models import User

# Clase 24

class Avatar(models.Model):

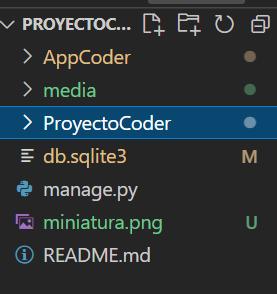
    user = models.ForeignKey(User, on\_delete=models.CASCADE)

    imagen = models.ImageField(upload\_to='avatares', null=True, blank = True)

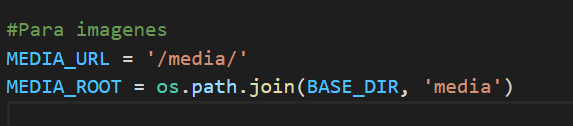
    def \_\_str\_\_(self):

        return f"{self.user} - {self.imagen}"

Es necesario crear una carpeta donde se guardarán todas las imágenes, por convención se la llama “media” y va en la carpeta raíz, por fuera del proyecto y de la app.



Luego hay que configurar la configuración de Django para que pueda acceder a estas imagenes, desde **settings.py**:

import os

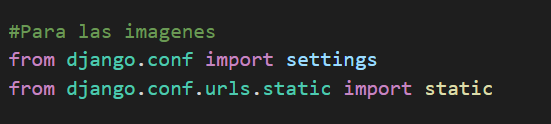
# Clase 24

MEDIA\_URL = '/media/'

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')

También debemos definir la urls para buscar las imágenes, desde **urls.py** del proyecto.

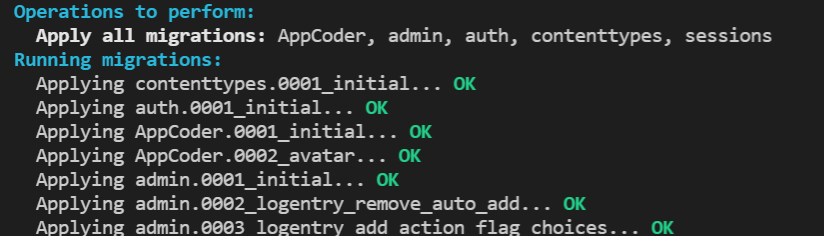
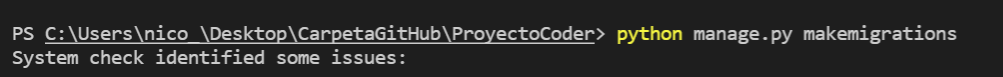
urlpatterns+= static(settings.MEDIA\_URL, document\_root = settings.MEDIA\_ROOT)



Al hacer un cambio en el modelo es importante volver a ejecutar:

**1- python manage.py makemigrations**

**2- python manage.py migrate**



Suele ser útil cuando uno genera cambios muy grandes en el modelo como borrar la base de datos y volverla a crear. En nuestro caso el cambio fue mínimo.

En el caso que en algún momento deban hacer eso, es necesario recordar que tendrán que generar otro superusuario

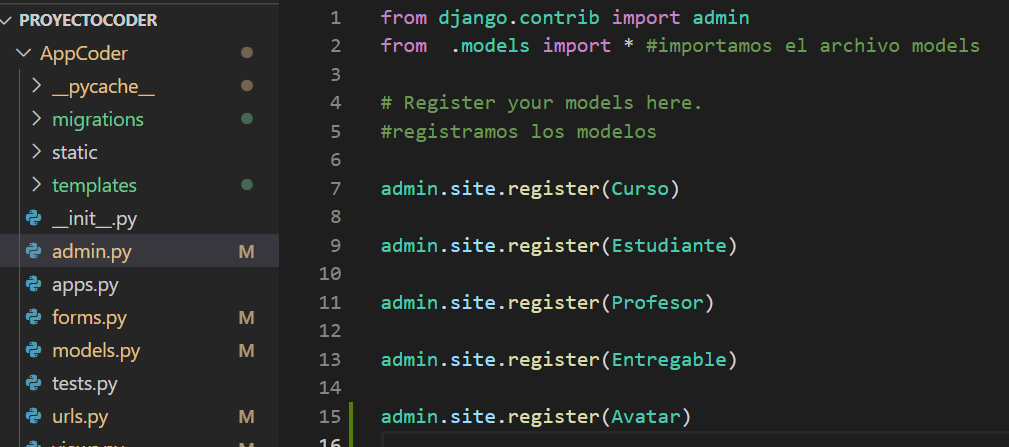
👇

**python manage.py createsuperuser**

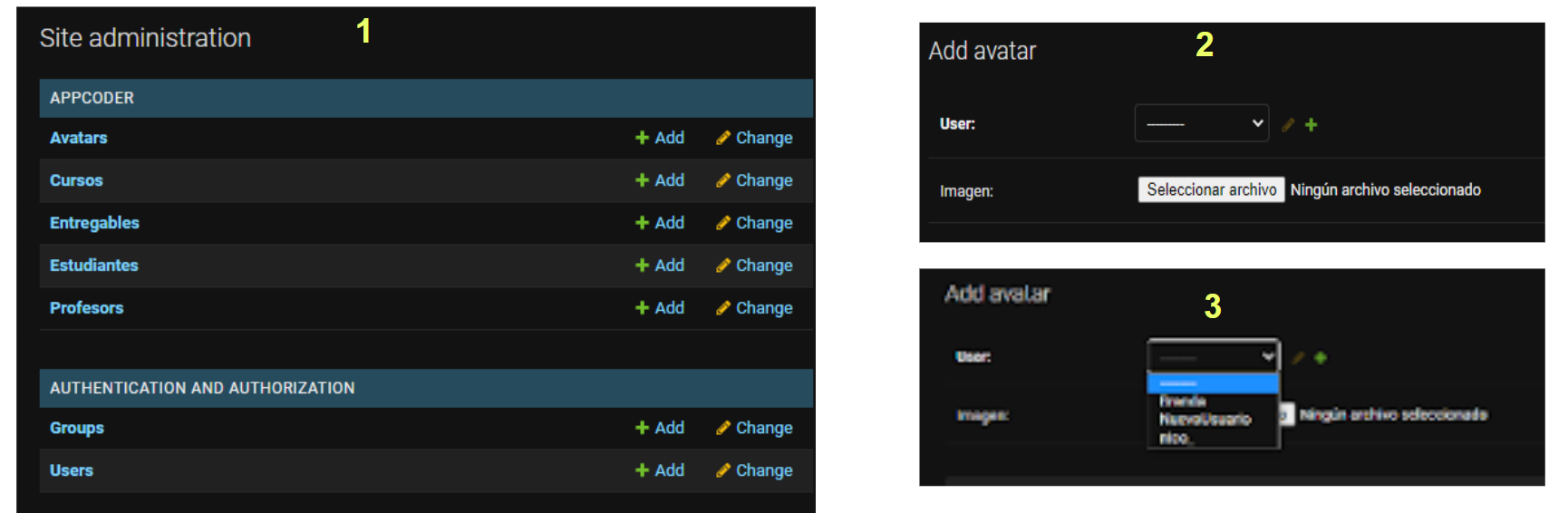
Ya casi estamos, solo resta permitir que se pueda subir una imagen.

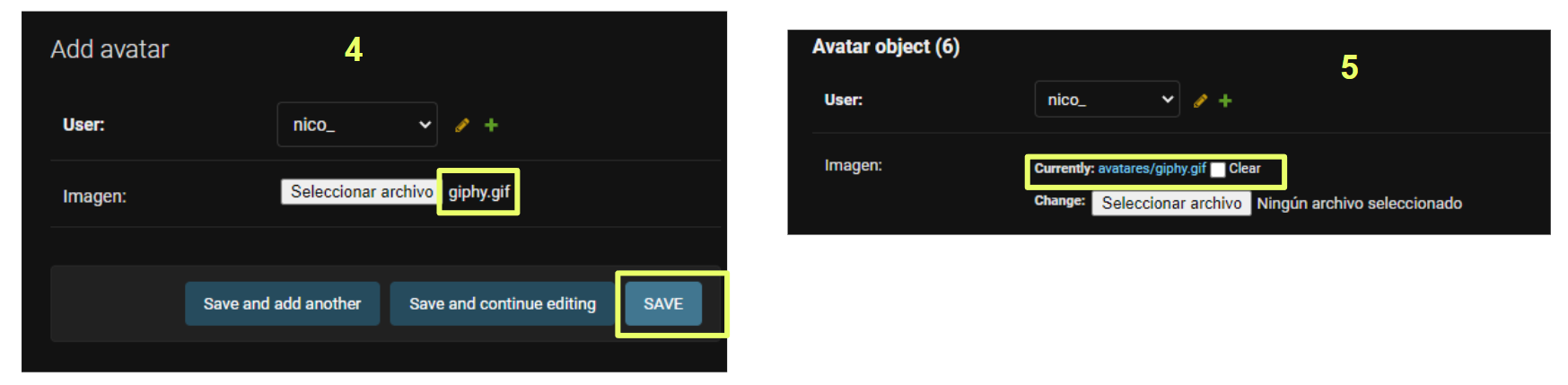
Para eso debemos dejar visible desde el panel de admin nuestro Avatar

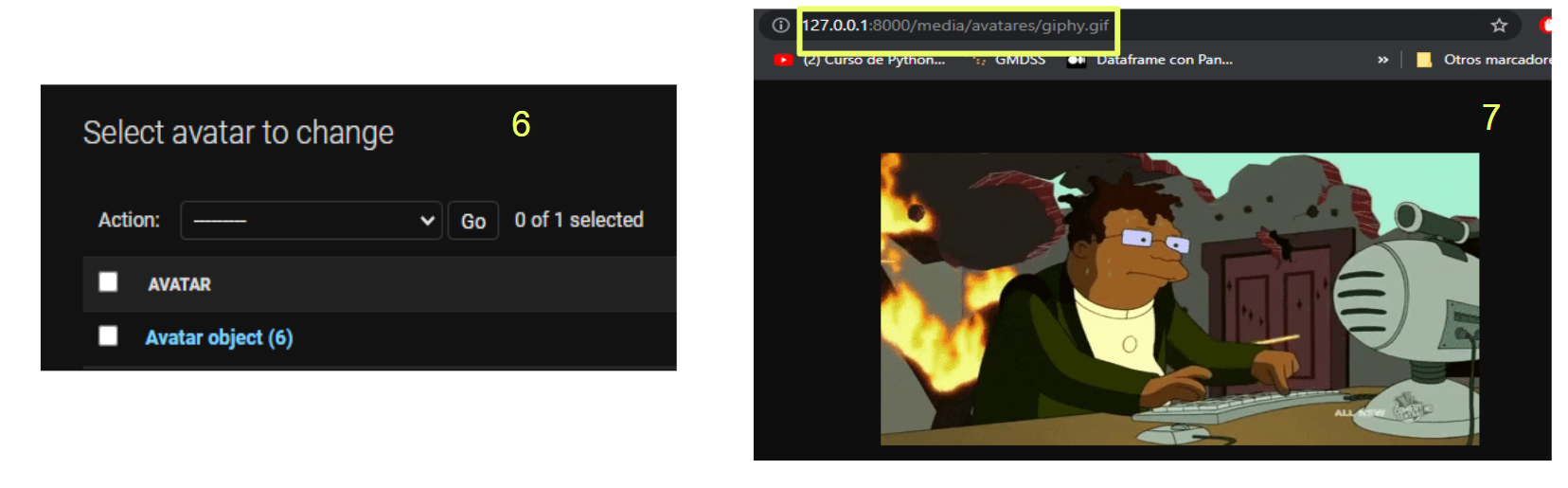
**admin.py**



Pasemos a la acción 👀



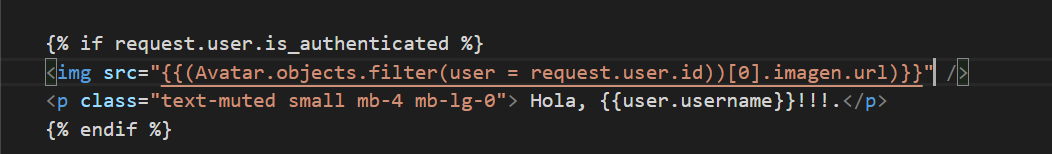




Ya sabemos subir imágenes e incluso que cada imagen se encuentre asociada a una determinada persona.

¿Cómo hacemos para mostrar esa imagen en nuestro avatar?

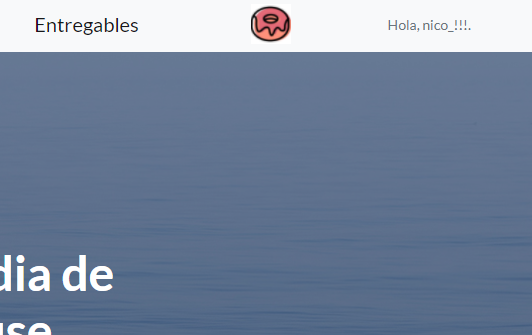
¡Es simple!, ¿no lo creen?, veámoslo 👀



{% if request.user.is\_authenticated %}

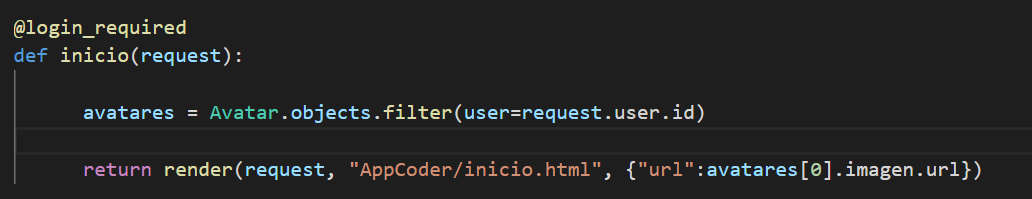
<img src=”{{(Avatar.ojects.filter(user=request.user.id))[0].imagen.url}}” />

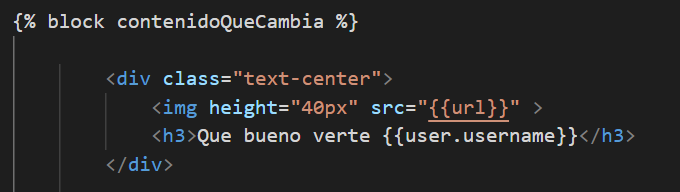
<p class=”text-muted small mb-4 mb-lg-0”> Hola, {{user.username}}!!!.</p>



O mejor aún podríamos enviar como un diccionario al contexto de cualquier html para ver esos datos

👇

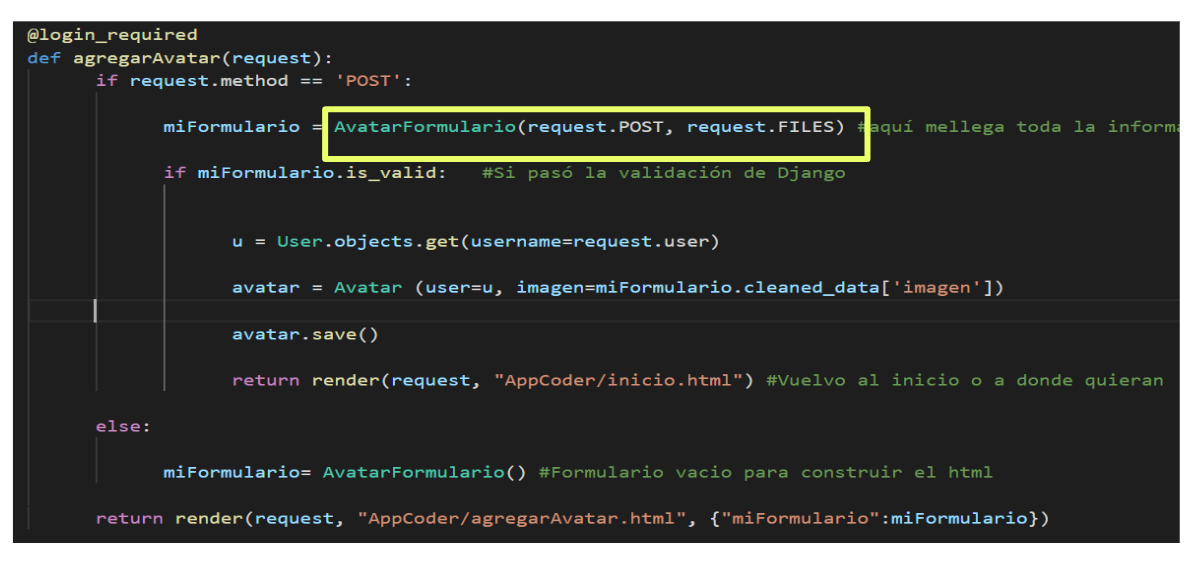




¡Se verá así!



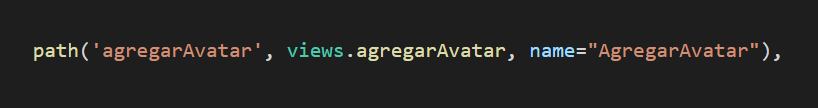
Por último, podríamos hacer un formulario que nos permita agregar un avatar pero sin la necesidad de ser miembro del staff, es decir un CRUD de imágenes.



El **agregarAvatar** queda así



A continuación la **urls.py**



**Unit test**

En castellano, Pruebas Unitarias, se refiere al método de comprobación de las “unidades” más pequeñas del software. Los componentes más pequeños que pueden probarse y cuyos resultados más significativos son los módulos.

Es recomendable comprobarlos en las primeras fases de desarrollo, pues en la fase de prueba, el módulo aún se puede corregir de forma relativamente rápida y poco costosa.

El desarrollador es quien ejecuta las pruebas y se encarga de corregir errores y asegurar la correcta funcionalidad de los componentes.

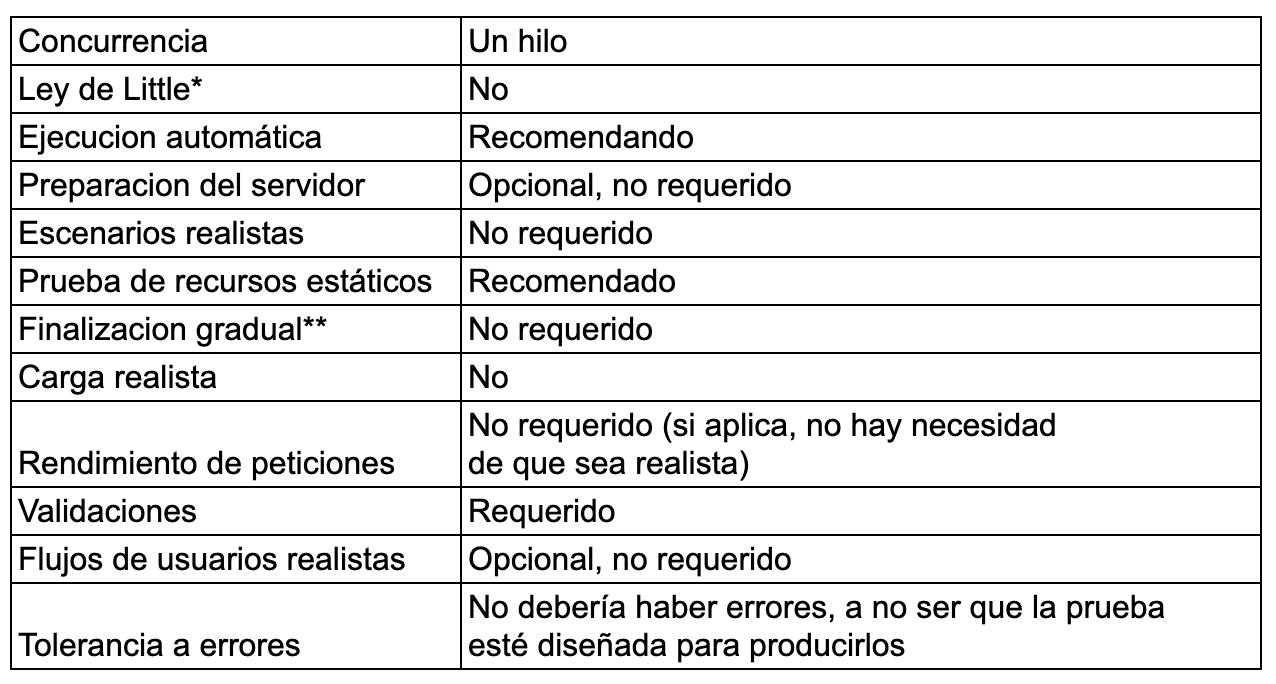
Es una tarea transversal a todo proyecto de programación por lo cual se realiza durante todo el desarrollo, desde el **Print(“Hola mundo”)** , hasta el final de las vistas más complejas con Django.

*Las pruebas unitarias consisten en que el desarrollador cree algunos métodos o funciones que exprimen al máximo al SW.*

Por ejemplo, cuando uno quiere probar que funciona bien el módulo de agregar Materia debería generar el nombre y la camada.

Una buena práctica o testeo Unitario sería generar camadas aleatorias con números negativos, positivos, repetidos, muy grandes o muy pequeños . De esta manera se comprueba si los datos se guardan correctamente o qué tipo de errores aparecen.

Lo mismo podríamos hacer con el nombre: podríamos generar nombres vacíos, nombres muy extensos, nombres con símbolos; para verificar que también se guardan bien.

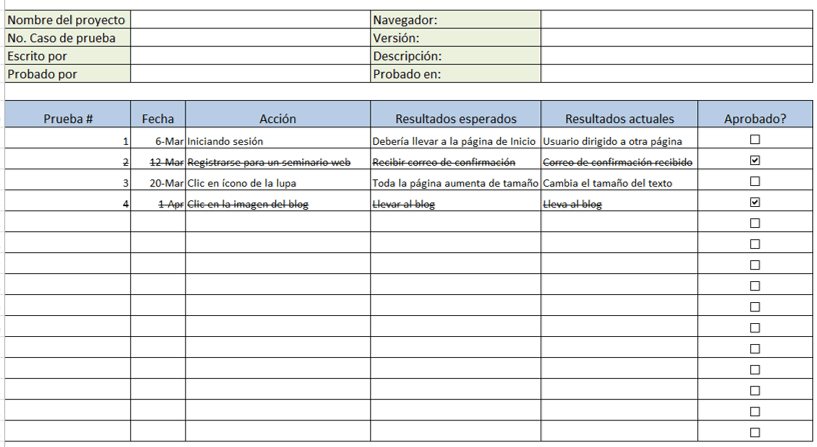


Una vez que se realizaron esas dos pruebas unitarias uno podría probar ambas en simultáneo e ir documentando en los llamados casos de prueba.

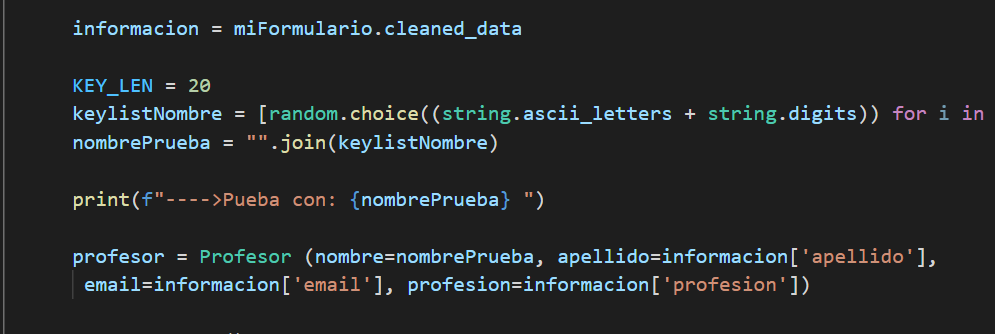
En estas plantillas se suelen ir documentando una por una las pruebas.

Luego todos los resultados obtenidos, o por lo menos los más importantes, se inmortalizan en una plantilla de casos de prueba.

Hay múltiples plantillas pero la idea es generar algo así 👇



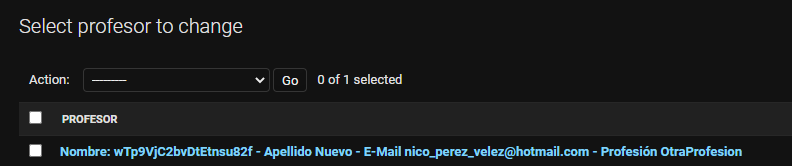
Haremos una prueba unitaria sobre “Crear Profesor” que funcionaba bien.



Intentemos guardar un nombre aleatorio de 20 caracteres (letras y números).



Así se vería el resultado de la ejecución.



La idea de estas pruebas no es que todo salga bien, todo lo contrario, la idea es encontrar errores para saber las limitaciones de nuestros productos.

Un trabajo que a nadie le da gusto hacerlo, pero que es necesario.

Siempre es mejor conocer los problemas antes que ignorarlos.