Esta clase va a ser

- grabada

Clase 24. PYTHON

Playground Avanzado Parte III

Temario

23	24	25
Playground Avanzado Parte II	**Playground Avanzado Parte III**	¿Y ahora qué?
✓ Login - Registro - Logout ✓ Mixin y Decoradores	✓ Edición de usuario ✓ Avatar ✓ Unit test	✓ Heroku - Pythonanywhere ✓ Pendientes de Python

Objetivos de la clase

- **Modificar** nuestro User.
- **Crear** un Avatar.
- **Ejecutar** guardado de imágenes en nuestra base de datos.

Repositorio Github

Te dejamos el acceso al Repositorio de Github donde encontrarás todo el material complementario y scripts de la clase.

✓ Repositorio Python

CODERHOUSE

Edición de usuario

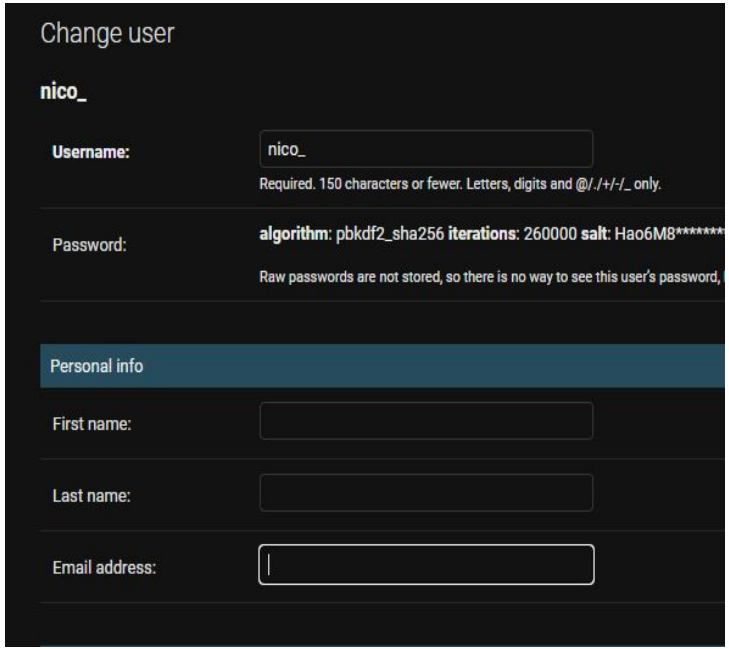
Edición de usuario

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.

CODERHOUSE

Edición de usuario



Recordemos que por defecto Django tiene los siguientes campos de usuario

👈

Edición de usuario

```
<!--Aquí va lo que cambia, y lo asociado a está vista :) -->
<h1>Formulario - Editar Perfil</h1>
<h1>De {{usuario}}</h1>

<form action="" method="POST">{% csrf_token %}

    <!-- Acá está la magia de Django-->

    <table>

        {{ miFormulario.as_table }}


    </table>


    <input type="submit", value="Enviar">

</form>
```

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

Edición de usuario

```python
@login_required
def editarPerfil(request):

    #Instancia del login
    usuario = request.user

    #Si es metodo POST hago lo mismo que el agregar
    if request.method == 'POST':
        miFormulario = UserEditForm(request.POST)
        if miFormulario.is_valid:   #Si pasó la validación de Django

            informacion = miFormulario.cleaned_data

            #Datos que se modificarán
            usuario.email = informacion['email']
            usuario.password1 = informacion['password1']
            usuario.password2 = informacion['password1']
            usuario.save()

            return render(request, "AppCoder/inicio.html") #Vuelvo al inicio o a donde quieran
    #En caso que no sea post
    else:
        #Creo el formulario con los datos que voy a modificar
        miFormulario= UserEditForm(initial={ 'email':usuario.email})

    #Voy al html que me permite editar
    return render(request, "AppCoder/editarPerfil.html", {"miFormulario":miFormulario, "usuario":usuario})
```

👈 Nuestra vista

Ver archivo: editarPerfil_views.txt

CODERHOUSE

Edición de usuario

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

👈 Urls

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

```
class UserEditForm(UserCreationForm):

    #Acá se definen las opciones que queres modificar del usuario,
    #Ponemos las básicas
    email = forms.EmailField(label="Modificar E-mail")
    password1 = forms.CharField(label='Contraseña', widget=forms.PasswordInput)
    password2 = forms.CharField(label='Repetir la contraseña', widget=forms.PasswordInput)


    class Meta:
        model = User
        fields = [ 'email', 'password1', 'password2']
        #Saca los mensajes de ayuda
        help_texts = {k:"" for k in fields}
```

👈 Form

Edición de usuario

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']
```

Para pensar

Ya sabemos cómo modificar determinados atributos del Usuario. Pero, **¿cómo haríamos para modificar aún más?**.O mejor dicho: **¿cómo haríamos para acceder a los otros datos del Usuario desde el registro?**

Contesta mediante el chat de Zoom

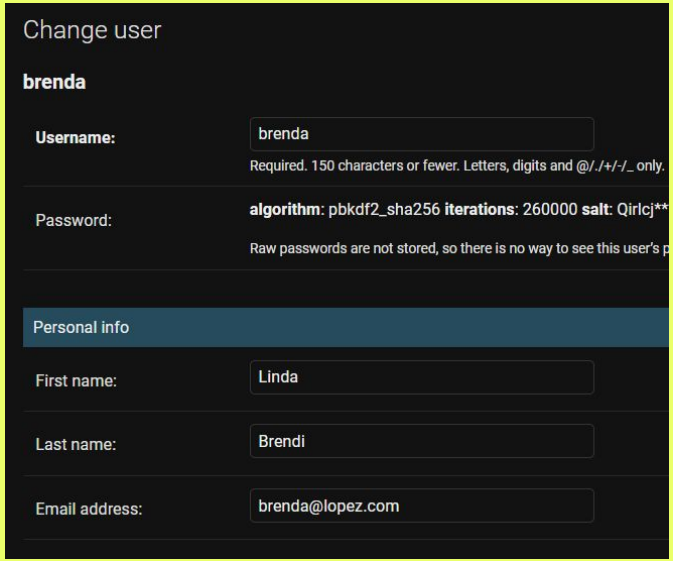
Ejemplo en vivo

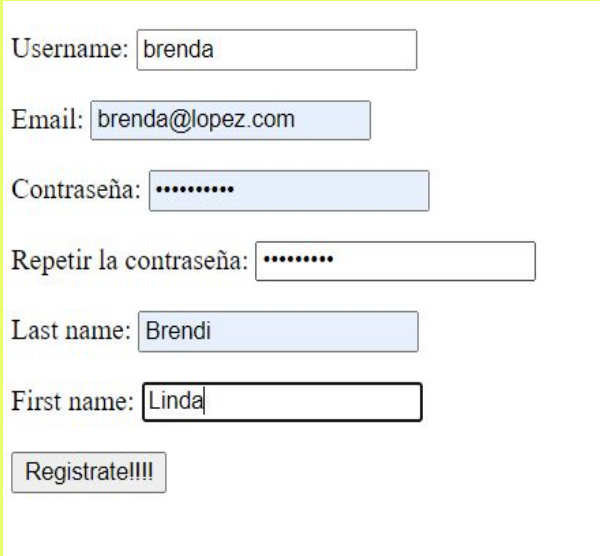
Veamos cómo modificar el registro de usuario para que además de sus datos básicos podamos agregar otros.

Solo modificando el forms.py

```python
class UserRegisterForm(UserCreationForm):

    username = forms.CharField()
    email = forms.EmailField()
    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 = ['username', 'email', 'password1', 'password2', 'last_name', 'first_name']
        #Saca los mensajes de ayuda
        help_texts = {k:"" for k in fields}
```

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





Avatar

Avatar

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

¿Cómo lo conseguimos?

CODERHOUSE

Avatar

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

```
<body>
    <!-- Navigation-->
    <nav class="navbar navbar-light bg-light static-top">
        <div class="container">
            <a class="navbar-brand" href="{% url 'Inicio' %}">Inicio</a>´
            <a class="navbar-brand" href="{% url 'Profesores' %}">Profesores</a>
            <a class="navbar-brand" href="{% url 'Cursos' %}">Cursos</a>
            <a class="navbar-brand" href="{% url 'Estudiantes' %}">Estudiantes</a>
            <a class="navbar-brand" href="{% url 'Entregables' %}">Entregables</a>
            <a class="btn btn-primary" href="#NADAAUN">INICIAR</a>
        </div>
    </nav>
```

👈 Recordemos lo que teníamos.

Avatar

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

```
<body>
    <!-- Navigation-->
    <nav class="navbar navbar-light bg-light static-top">
        <div class="container">
            <a class="navbar-brand" href="{% url 'Inicio' %}">Inicio</a>´
            <a class="navbar-brand" href="{% url 'Profesores' %}">Profesores</a>
            <a class="navbar-brand" href="{% url 'Cursos' %}">Cursos</a>
            <a class="navbar-brand" href="{% url 'Estudiantes' %}">Estudiantes</a>
            <a class="navbar-brand" href="{% url 'Entregables' %}">Entregables</a>
            <a class="btn btn-primary" href="{% url 'Loguin' %}">INICIAR</a>
        </div>
    </nav>
```

Avatar

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

```
<a class="navbar-brand" href="{% url 'Entregables' %}">Entregables</a>
{% if not request.user.is_authenticated %}
<a class="btn btn-primary" href="{% url 'Login' %}">INICIAR</a>
{% endif %}
```

CODERHOUSE

Avatar

```
{% if not request.user.is_authenticated %}
<a class="btn btn-primary" href="{% url 'Login' %}">INICIAR</a>
{% endif %}

{% if request.user.is_authenticated %}
<p class="text-muted small mb-4 mb-lg-0"> Hola, {{user.username}}!!!.</p>
{% endif %}
```

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

👈 Pero si ya está logueada podemos mostrar su info.

CODERHOUSE

Avatar

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.
👇

```python
class Avatar(models.Model):
    #vinvulo con el usuario
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    #Subcaperta avatares de media :)
    imagen = models.ImageField(upload_to='avatares', null=True, blank = True)
```

Avatar

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

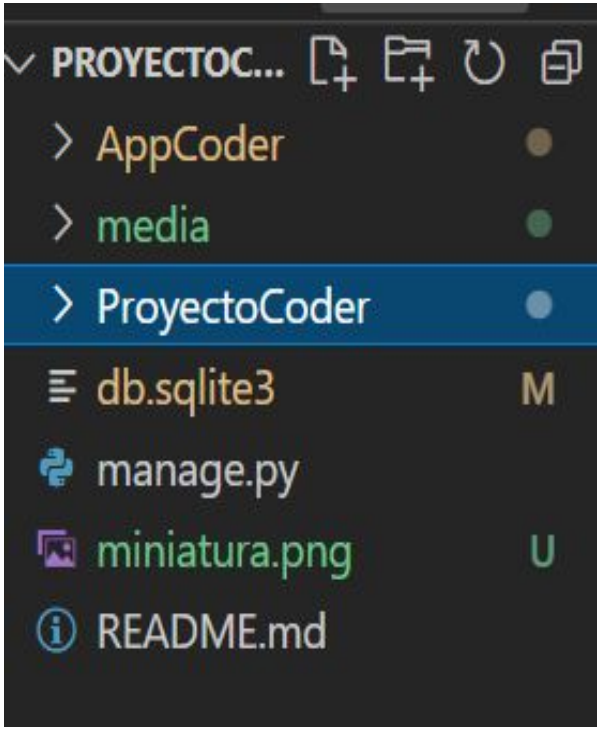
```python
class Avatar(models.Model):
    #vinvulo con el usuario
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    #Subcaperta avatares de media :)
    imagen = models.ImageField(upload_to='avatares', null=True, blank = True)
```

Avatar

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}"
```

Avatar



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.

Avatar

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

```
#Para imagenes
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

```
import os


# Clase 24
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

Avatar

```
#Para las imagenes
from django.conf import settings
from django.conf.urls.static import static
```

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

```
urlpatterns+=
static(settings.MEDIA_URL,
document_root = settings.MEDIA_ROOT)
```

```
urlpatterns = [
    path('admin/', admin.site.urls),

    path('AppCoder/', include('AppCoder.urls')),

]


#Para las imagenes
urlpatterns+= static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

CODERHOUSE

Avatar

```
PS C:\Users\nico \Desktop\CarpetaGitHub\ProyectoCoder> python manage.py makemigrations
System check identified some issues:
```

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

1- python manage.py makemigrations
2- python manage.py migrate

```
Operations to perform:
  Apply all migrations: AppCoder, admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying AppCoder.0001_initial... OK
  Applying AppCoder.0002_avatar... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
```

Avatar

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
```

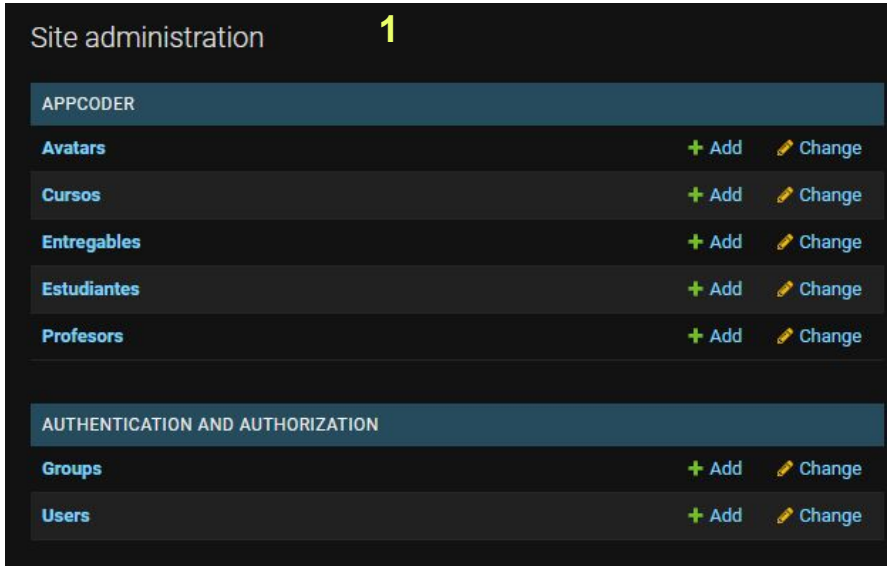
Avatar

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

Avatar

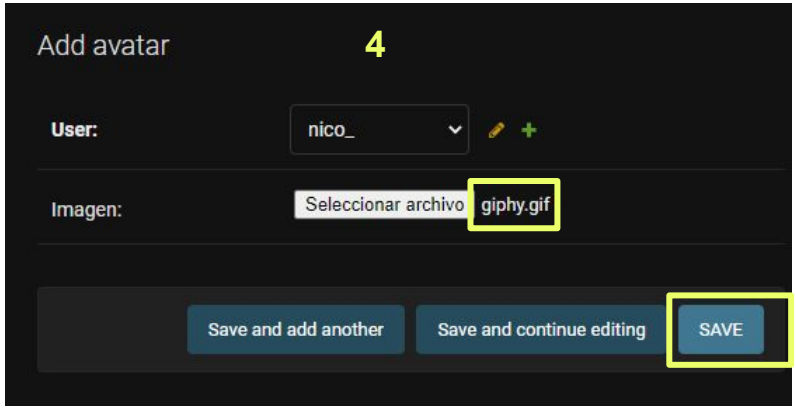
Pasemos a la acción 👀

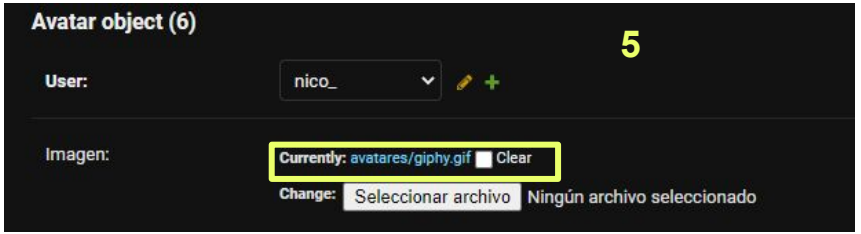


CODERHOUSE

Avatar

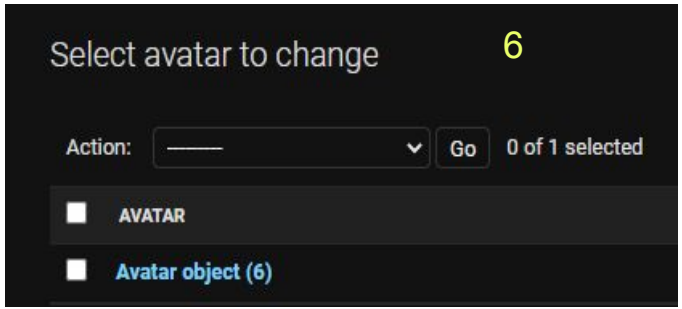
Pasemos a la acción 👀





Avatar

Pasemos a la acción 👀



Avatar

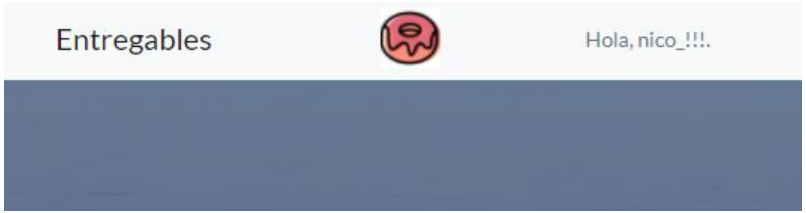
Ya sabemos subir imágenes e incluso que cada imagen se encuentre asociada a una determinada persona.

¿Cómo hacemos para mostrar esa imagen en nuestro avatar?

Avatar

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

```
{% if request.user.is_authenticated %}
<img src="{{(Avatar.objects.filter(user = request.user.id))[0].imagen.url)}}" />
<p class="text-muted small mb-4 mb-lg-0"> Hola, {{user.username}}!!!.</p>
{% endif %}
```



CODERHOUSE

Avatar

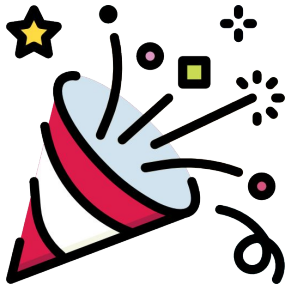
O mejor aún podríamos enviar como un diccionario al contexto de cualquier html para ver esos datos

👇

```
@login_required
def inicio(request):

    avatares = Avatar.objects.filter(user=request.user.id)

    return render(request, "AppCoder/inicio.html", {"url":avatares[0].imagen.url})
```

Avatar

¡Se verá así!



```
{% block contenidoQueCambia %}

    <div class="text-center">
        <img height="40px" src="{{url}}" >
        <h3>Que bueno verte {{user.username}}</h3>
    </div>
```

Avatar

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.

```
@login_required
def agregarAvatar(request):
      if request.method == 'POST':

            miFormulario = AvatarFormulario(request.POST, request.FILES) #aquí mellega toda la informa

            if miFormulario.is_valid:   #Si pasó la validación de Django


                  u = User.objects.get(username=request.user)

                  avatar = Avatar (user=u, imagen=miFormulario.cleaned_data['imagen'])

                  avatar.save()

                  return render(request, "AppCoder/inicio.html") #Vuelvo al inicio o a donde quieran

      else:

            miFormulario= AvatarFormulario() #Formulario vacio para construir el html

      return render(request, "AppCoder/agregarAvatar.html", {"miFormulario":miFormulario})
```

Avatar

El **agregarAvatar** queda así

```
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Cargar Avatar</title>
</head>
<body>

    <h3>Cargar Avatar</h3>
    <form method="POST" enctype="multipart/form-data">

        {% csrf_token %}
        {{ miFormulario.as_p  }}


        <input type="submit" value="Actualizar">


    </form>
```

Avatar

A continuación la **urls.py**

```
path('agregarAvatar', views.agregarAvatar, name="AgregarAvatar"),
```

Agregar una imagen

Elegir cualquier clase de tu modelo y agregarle un dato del tipo imagen, lograr hacer un alta de imagen.

Duración: **15 minutos**

CODERHOUSE

ACTIVIDAD EN CLASE

Agregar una imagen

Pensando en tu entrega final, elige alguna de tus clases del modelo que tengas y agrégale un atributo imagen, trata de hacer un alta de esa imagen que se vea reflejada en la BD.

Break

¡10 minutos y volvemos!

Unit test

Unit test

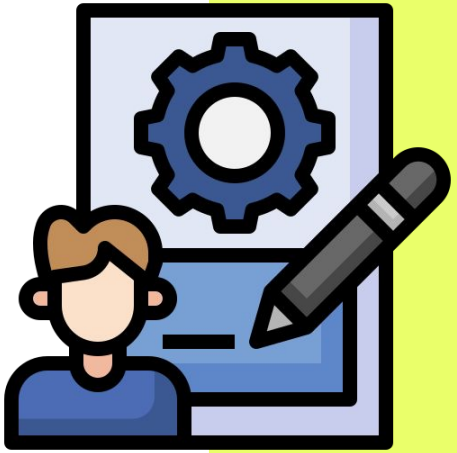
En castellano, **Pruebas Unitarias**, se refiere al método de comprobación de las “unidades” más pequeñas del software. Los componentes más pequeños que pueden probarse y cuyos resultados más significativos son los módulos.

Unit test

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.

Unit test

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.



CODERHOUSE

Unit test

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.

Unit test

Concurrencia	Un hilo
Ley de Little*	No
Ejecucion automática	Recomendando
Preparacion del servidor	Opcional, no requerido
Escenarios realistas	No requerido
Prueba de recursos estáticos	Recomendado
Finalizacion gradual**	No requerido
Carga realista	No
Rendimiento de peticiones	No requerido (si aplica, no hay necesidad de que sea realista)
Validaciones	Requerido
Flujos de usuarios realistas	Opcional, no requerido
Tolerancia a errores	No debería haber errores, a no ser que la prueba esté diseñada para producirlos

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

Unit test

Concurrencia	Un hilo
Ley de Little*	No
Ejecucion automática	Recomendando
Preparacion del servidor	Opcional, no requerido
Escenarios realistas	No requerido
Prueba de recursos estáticos	Recomendado
Finalizacion gradual**	No requerido
Carga realista	No
Rendimiento de peticiones	No requerido (si aplica, no hay necesidad de que sea realista)
Validaciones	Requerido
Flujos de usuarios realistas	Opcional, no requerido
Tolerancia a errores	No debería haber errores, a no ser que la prueba esté diseñada para producirlos

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.

Unit test

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 👇

Nombre del proyecto		Navegador:	
No. Caso de prueba		Versión:	
Escrito por		Descripción:	
Probado por		Probado en:	

Prueba #	Fecha	Acción	Resultados esperados	Resultados actuales	Aprobado?
1	6-Mar	Iniciando sesión	Debería llevar a la página de Inicio	Usuario dirigido a otra página	☐
2	~~12-Mar~~	~~Registrarse para un seminario web~~	~~Recibir correo de confirmación~~	~~Correo de confirmación recibido~~	☑
3	20-Mar	Clic en ícono de la lupa	Toda la página aumenta de tamaño	Cambia el tamaño del texto	☐
4	~~1-Apr~~	~~Clic en la imagen del blog~~	~~Llevar al blog~~	~~Lleva al blog~~	☑
					☐

Ejemplo en vivo

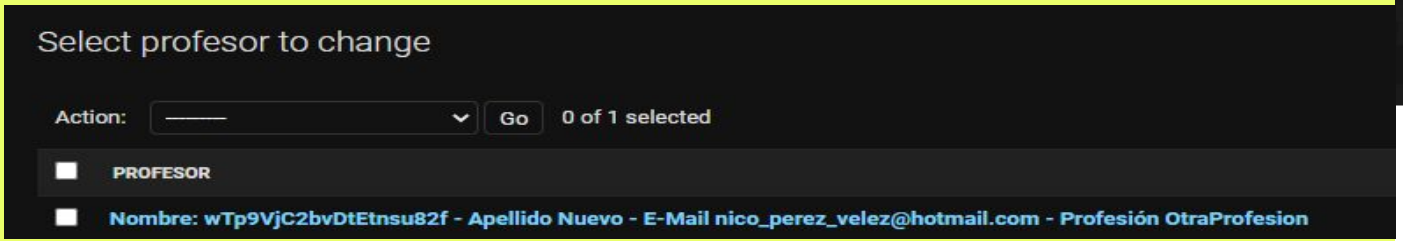
Veamos un ejemplo sencillo para entender el problema.

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

```
informacion = miFormulario.cleaned_data

KEY_LEN = 20
keylistNombre = [random.choice((string.ascii_letters + string.digits)) for i in
nombrePrueba = "".join(keylistNombre)

print(f"---->Pueba con: {nombrePrueba} ")

profesor = Profesor (nombre=nombrePrueba, apellido=informacion['apellido'],
 email=informacion['email'], profesion=informacion['profesion'])
```

```
30" required id="id_profesion"></td></tr>
---->Pueba con: wTp9VjC2bvDtEtnsu82f
```

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



Así se vería el resultado de la ejecución.

PARA RECORDAR

Unit test

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.

Documento casos de prueba

Documentar el caso de prueba anterior.

Duración: **10 minutos**

CODERHOUSE

ACTIVIDAD EN CLASE

Documento casos de prueba

Dado el caso de prueba de la presentación, documentaro individualmente y luego genera la plantilla de caso de prueba, para ello se debe utilizar el archivo: "***Casos de prueba.xlsx***"

CODERHOUSE

¿Preguntas?

Opina y valora esta clase

Muchas gracias.

#DemocratizandoLaEducación

Resumen
de la clase hoy

- ✓ Edición de usuario.
- ✓ Avatar.
- ✓ Uni Test.