Proyecto Final de Python

Comisión 18695

Francisco Martín Carranza

Edición del avatar propio del usuario

Se plantea la necesidad de permitir que un usuario **únicamente logueado** tenga la oportunidad de cargar su propio avatar o modificarlo si así lo desea. En este caso no se implementan requisitos al archivo del avatar (por ejemplo, tipo de formato o máximo espacio ocupado por la imagen), pero sí debe ser una imagen.

Primero se presenta la función completa de **views.py**, y luego se detalla cada sección:

```
@login_required
def editar_avatar(request):
    user_avatar = Avatar.objects.filter (user=request.user.id)
    usuario = request.user

    if request.method == 'POST':
        form = AvatarForm(request.POST, request.FILES)

        if form.is_valid():
            try:
                avatar = Avatar.objects.get(user=usuario)
                avatar.avatar = form.cleaned_data['avatar']
            except:
                avatar = Avatar(user=usuario, avatar=form.cleaned_data['avatar'])
            avatar.save()
            return render (request, 'AppCoder/index.html', {'flag': True, 'mensaje':
'Avatar cargado con éxito!'})
    form = AvatarForm()
    try:
        return render(request, 'AppCoder/editar_avatar.html', {'form': form, 'mensaje':
'', "url": user_avatar[0].avatar.url})
    except:
        return render(request, 'AppCoder/editar_avatar.html', {'form': form, 'mensaje':
'Estás a punto de cargar tu primer Avatar!'})
```

Se utiliza el **decorator @login_required**, cuya función es obligar al usuario a estar logueado para ingresar a esta vista. Además, se filtra y obtiene desde la base de datos el avatar y usuario que está iniciando la solicitud:

```
@login_required
def editar_avatar(request):
    user_avatar = Avatar.objects.filter (user=request.user.id)
    usuario = request.user
```

Si el método es **POST** y el formulario es válido, es decir, si la carga del avatar es correcta intentará encontrar campo avatar que corresponde al usuario que está iniciando la solicitud. Esto significa que ya hay un avatar cargado anteriormente.
Si no existe un avatar creado por el usuario (es decir, que va a cagar la imagen por primera vez), se dará una excepción. Entonces, con la ayuda de **try** vs **except** se almacenará el avatar en un usuario aún que no tiene ese campo en la base de datos. Finalmente, se da un **.save()** para guardarlo en la base de datos con **flag** para mostrar el mensaje de "Avatar cargado con éxito!" en el **Home**:

```
if request.method == 'POST':
        form = AvatarForm(request.POST, request.FILES)

        if form.is_valid():
            try:
                avatar = Avatar.objects.get(user=usuario)
                avatar.avatar = form.cleaned_data['avatar']
            except:
                avatar = Avatar(user=usuario,
avatar=form.cleaned_data['avatar'])
            avatar.save()
            return render (request, 'AppCoder/index.html', {'flag': True,
'mensaje': 'Avatar cargado con éxito!'})
```

Si el método no es **POST** significa que el usuario está apenas ingresando a la URL. Se envía el formulario vacío y luego intentará mostrar el avatar del usuario (claro, siempre y cuando ya existiera uno). Esto se hace apuntando al primer lugar (**[0]**) de la variable **user_avatar**. Caso contrario, se dará la excepción **except** y un **mensaje** indicando que está por cargarse el primer avatar:

```
    form = AvatarForm()
    try:
        return render(request, 'AppCoder/editar_avatar.html', {'form':
form, 'mensaje': '', "url": user_avatar[0].avatar.url})
    except:
        return render(request, 'AppCoder/editar_avatar.html', {'form':
form, 'mensaje': 'Estás a punto de cargar tu primer Avatar!'})
```

A continuación el formulario de edición del avatar, donde se carga el **.ImageField()**:

```
class AvatarForm(forms.Form):
    avatar = forms.ImageField()
```

El modelo **Avatar** para la base de datos, con **models.CASCADE** indicando que en caso de borrar el usuario se borrará también el avatar asociado:

```
class Avatar(models.Model):
    user = models.ForeignKey(User, on_delete= models.CASCADE)
    avatar = models.ImageField(upload_to= 'avatares', null= True, blank=
True)
    def __str__(self):
         return f'{self.user}'
```

En cuanto a **index.html**, el **flag** indicará el mensaje de “Avatar cargado con éxito!”:

```
        {% if flag %}
            <h3 style = color:blueviolet>{{mensaje}}</h3>
        {% endif %}
        {% endblock body_block %}
```

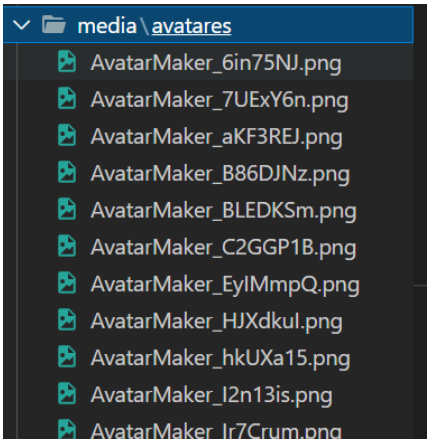
En cuanto a dónde se almacenan los avatares cargados, esto se realiza en la carpeta ‘**/media**’ gracias a las siguientes configuraciones de **settings.py** del Proyecto general (es decir, no desde la aplicación creada):

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

Y desde **urls.py** del proyecto:

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

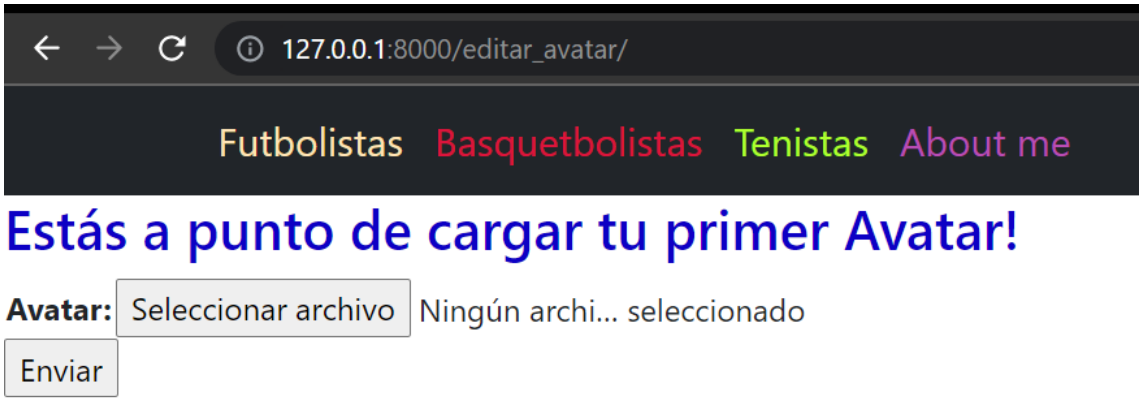
Se recuerda que en la **class Avatar** ya enunciada anteriormente, el campo **upload_to** indica que se cargarán entonces en ‘**/media/avatares**’:



Vale aclarar también que todo esto está preparado para realizarse también desde la dirección URL http://127.0.0.1:8000/admin/ o donde corresponda según la IP y puerto del servidor montado.

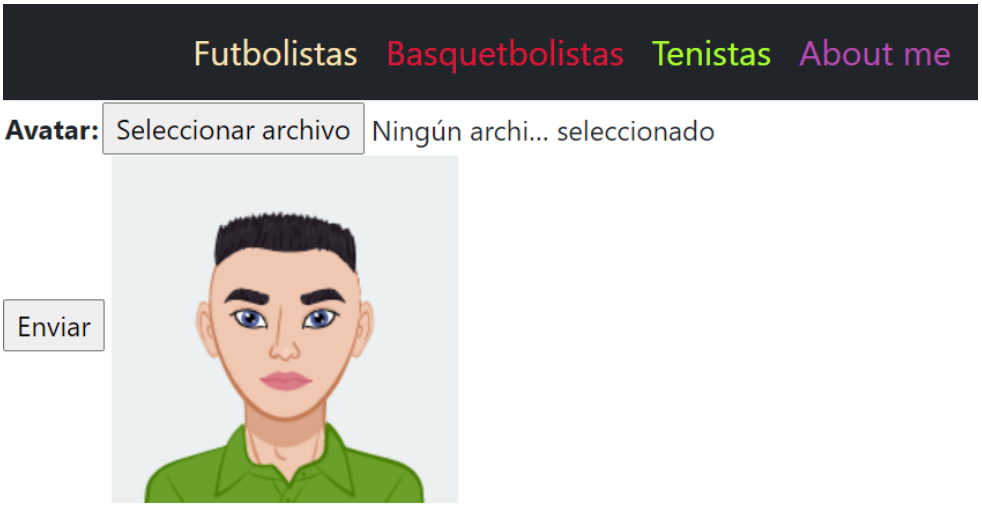
Ejemplos:

1. Ingreso por primera vez a la URL:



2. Carga exitosa del avatar:

3. Edición del avatar, donde se muestra el ya cargado anteriormente:



4. Vista del nuevo avatar cargado, que fue modificado exitosamente: