

## 1.Inserción en Django

Para poder incorporar un formulario en Django, igualmente hay que crear una vista con la página que se encargará de realizar el insert.

Siguiendo con nuestros ejemplos, vamos a proceder a incorporar la posibilidad de poder introducir un nuevo autor, pues hasta ahora sólo los podíamos introducir por el admin. En condiciones normales, al admin sólo van a poder tener acceso super usuarios.

Incorporamos un enlace en nuestra página autor.html que se dirija a nuestra vista dónde realizaremos la llamada al ORM que haga la inserción.

```
{% extends 'index.html' %}
{%block title%} Listado de Autores {%endblock title%}

{%block content%}
<ul>
    {% for aut in authors %}
        <li>{{ aut }}</li>
        <li>{{ aut.name }}</li>      <!-- Lo que contiene la variable autor es una lista -->
    {% endfor %}
</ul>
<div><a href="">Registrar nuevo autor</a></div>
{%endblock content%}
```

Dentro de href, tendremos una url que debe estar asociada a una vista.

Dentro de views.py añadimos esa nueva vista:

```
def autor_create (request):
    return render(request, 'create_autor.html') #Si tenemos las páginas dentro de alguna
carpeta dentro de template, entonces habría que especificar el nombre de la carpeta Ej:
return render(request, 'nombre_carpeta/create_autor.html')
```

Siguiente paso sería irnos al fichero urls.py y añadir la nueva función con el path que nos interese. Recordar que hay que importar la nueva función o directamente poner un \*

```
urlpatterns = [
    #path('hola/', views.index), #indicamos que la vista la queremos mostrar en esa ruta.
    path('', home), #quí no habría que meter la carpeta

    #path('otramas/', views.home),
    path('admin/', admin.site.urls),
    path('autor/', autor_list),
    path('libro/', libro_list),
    path ('new-autor/', autor_create),
]
```

Ahora en nuestra página autor.html dónde habíamos incluido el href, ya podemos incorporar el enlace dónde se tiene que ir:

```
<div><a href="/new-autor/">Registrar nuevo autor</a></div>
```

Obviamente hay que crear la nueva página web que hemos dicho que se llamará 'create\_autor.html'

Hay que crearla igualmente que extienda de index.html para seguir usando el formato de plantillas, con el nuevo título:

```
{% extends 'index.html' %}
{%block title%} Registrar nuevo autor {%endblock title%}

{%block content%}

{%endblock content%}
```

Podríamos meter dentro del bloque content, el típico formulario html con todos los campos y con el action correspondiente, pero Django intenta ahorrarnos todo ello, utilizando los formularios de Django. Estos formularios van a ser una clase que van a tomar como referencia nuestro modelo, nuestras tablas. En estos modelos ya teníamos definidos cada uno de los campos así como las especificaciones, con lo que nos puede ayudar con las validaciones.

Para comenzar hay que crear dentro de nuestra aplicación un nuevo archivo que por convención se llamará forms.py. En este .py incluiremos el siguiente código.

```
from django import forms
from core.models import Author # importamos el modelo que nos hace falta, en este caso
                                # necesitamos insertar autores

#Creamos nuestro primer Form

class AutorForm (forms.ModelForm): #Vamos a crear un formulario de Django , basado en un
    # modelo, por eso hacemos la herencia
    # Los forms de django tienen su clase meta, dónde indicamos el modelo al cual hacemos
    # referencia.
    # Con estas dos clases lo que estamos diciendo es que Django debe crearnos un form que haga
    # referencia el modelo author
    class Meta:
        model=Author
        # Luego hay que decirle qué campos queremos que tome de este modelo. A veces el modelo
        # tiene muchos campos pero sin embargo no hace falta introducirlos todos. Podemos colocar una
        # lista de los campos que aparezcan. También existe la opción de poner '__all__',
        # # fields= ('name','last_name') #En esta caso especificamos los campos que queramos que
        # aparezcan en el form.
        # En nuestro caso como solo tenemos tres campos vamos a ponerlos todos:
        fields='__all__'
```

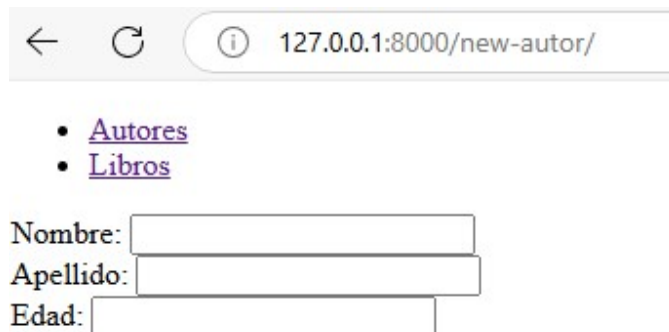
Una vez que cree el form, voy a importarlo en views.py añadiéndolo las siguientes líneas añadiendo además a la función autor\_create el contexto, pasándolo como parámetro:

```
.....  
  
from core.forms import AutorForm  
  
.....  
  
def autor_create (request):  
    return render(request, 'create_autor.html', {'autor_form': AutorForm}) #Si tenemos las  
    páginas dentro de alguna carpeta dentro de template, entonces habría que especificar el  
    nombre de la carpeta Ej: return render(request, 'nombre_carpeta/create_autor.html')  
    #Hemos añadido el contexto para pasarlo como parámetro a la página create_autor.html
```

Nos iremos ahora a nuestra página create\_autor.html, y pintaremos simplemente la variable que le hemos mandado desde views.py, autor\_form:

```
{% extends 'index.html' %}  
{% block title%} Registrar nuevo autor {%endblock title%}  
  
{% block content%}  
{{autor_form}}  
{%endblock content%}
```

Recargando nos visualizará los campos indicados.



← ↻ ⓘ 127.0.0.1:8000/new-autor/

- [Autores](#)
- [Libros](#)

Nombre:

Apellido:

Edad:

Hasta ahora hemos pintado el formulario pero necesitamos poder guardar lo que se escribe ahí. Para ello debemos encerrar la variable que pintamos dentro de un bloque form de html e indicarle el action así como el botón submit:

```
{% extends 'index.html' %}
{% block title%}  Registrar nuevo autor {%endblock title%}

{%block content%}
<form action="">
    {{autor_form}}
    <button type="submit">Guardar</button>
</form>

{%endblock content%}
```

Si le damos al botón Guardar sin introducir nada, nos va a avisar automáticamente que ese campo es obligatorio, pues en el modelo así se especifica:



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8000/new-autor/'. The page content includes a navigation menu with links for 'Autores' and 'Libros'. Below the menu is a form titled 'Registrar nuevo autor'. The form contains three input fields: 'Nombre:', 'Apellido:', and 'Edad:'. The 'Edad:' field is highlighted with a red error message that says 'Rellene este campo.' (Fill in this field). Below the fields is a 'Guardar' (Save) button.

Si el action lo dejamos vacío, va a utilizar la misma url para enviar, pues por defecto el método es get, y los datos introducidos a aparecerán por parámetro. En nuestro caso, el método debe ser POST.

```
{% extends 'index.html' %}
{% block title%}  Registrar nuevo autor {%endblock title%}

{%block content%}
<form action="" method="POST">
    {{autor_form}}
    <button type="submit">Guardar</button>
</form>

{%endblock content%}
```

Tal cual está ahora mismo, el resultado de guardar nos da el siguiente error:

**Forbidden** (403)

CSRF verification failed. Request aborted.

**Help**Reason given for failure:  
CSRF token missing.In general, this can occur when there is a genuine Cross Site Request Forgery, or when [Django's CSRF mechanism](#) has not been used correctly. For POST forms, you need to ensure:

- Your browser is accepting cookies.
- The view function passes a request to the template's [render](#) method.
- In the template, there is a `{% csrf_token %}` template tag inside each POST form that targets an internal URL.
- If you are not using `CsrfViewMiddleware`, then you must use `csrf_protect` on any views that use the `csrf_token` template tag, as well as those that accept the POST data.
- The form has a valid CSRF token. After logging in in another browser tab or hitting the back button after a login, you may need to reload the page with the form, because the token is rotated after a login.

You're seeing the help section of this page because you have `DEBUG = True` in your Django settings file. Change that to `False`, and only the initial error message will be displayed.You can customize this page using the `CSRF_FAILURE_VIEW` setting.

Hay que incorporar un código en los formularios, para que Django sepa que el formulario tiene una identificación única y que la información que se envía es verídica. Es un código necesario por seguridad.

```
{% extends 'index.html' %}
{% block title%}  Registrar nuevo autor {%endblock title%}

{%block content%}
<form action="" method="POST">
    {%csrf_token%}
    {{autor_form}}
    <button type="submit">Guardar</button>
</form>

{%endblock content%}
```

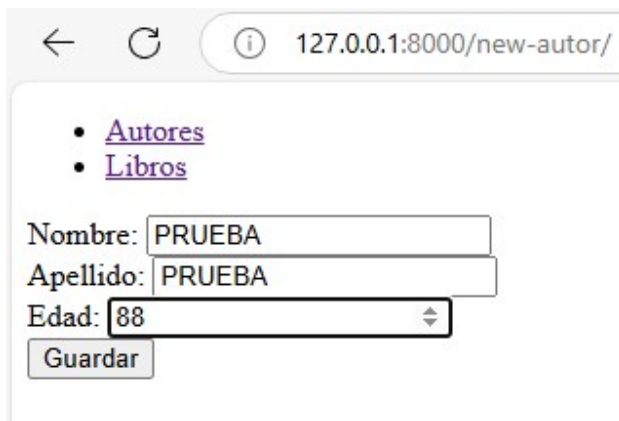
Si se inspecciona el código html generado de la página, este token aparecerá como campo hidden, con un valor de un token, y Django lo va a dar por válido.

```
<input type="hidden" name="csrfmiddlewaretoken"
value="DIXUXzIQEjj4ztb8jVfEOXBG9vhxvLrf1g3l7tpME0Vx1ELcbP1MAQSowcvJxVje">
```

El name es ese pues dentro de nuestro setting, en la sección **MIDDLEWARE** hay un middleware que hace referencia al csrf

En nuestra vista, la función `autor_create` hay que modificarla para que distinga entre los métodos desde los cuales se le llama. De tal manera que si el método es get, el código sería el que hay y si el método es post sería el siguiente:

```
def autor_create (request):
    if request.method == 'GET':
        return render(request, 'create_autor.html', {'autor_form': AutorForm}) #Si tenemos las páginas
        dentro de alguna carpeta dentro de template, entonces habría que especificar el nombre de la
        carpeta return Ej:render(request, 'nombre_carpeta/create_autor.html')
        #Hemos añadido el contexto para pasarlo como parámetro a la página create_autor.html
    if request.method == 'POST':
        #Estaremos aquí en esta opción cuando le damos a guardar. Para obtener la información que
        envía el formulario:
        # EN request.POST es dónde tenemos toda esa información. De hecho si imprimimos esa variable la
        podemos ver.
        form=AutorForm(data=request.POST)
        #Hemos creado una instancia de un form de Django al que le estamos pasando los datos usando la
        variable data, de esta manera entiende que se va a registrar una información (que va a ser un
        diccionario)
        if form.is_valid: #Realizará de forma automática las validaciones que se han establecido en
        el modelo
            form.save() #Guarda en la BD. En pocas líneas hemos validado y guardado.
            return redirect ('/autor/') #Hay que importar el redirect en shortcuts. Debe redirigir
            al listado de autores.
        else:
            #Debo volver a crear una instancia del form para dar la opción de poder escribir otra vez
            los datos, con los mismos datos que ha introducido para que vea qué cual puede ser el error.
            form=AutorForm(data=request.POST)
            return render (request, 'create_autor.html',{'autor_form': AutorForm}) #Es como si fuera un
            GET
```



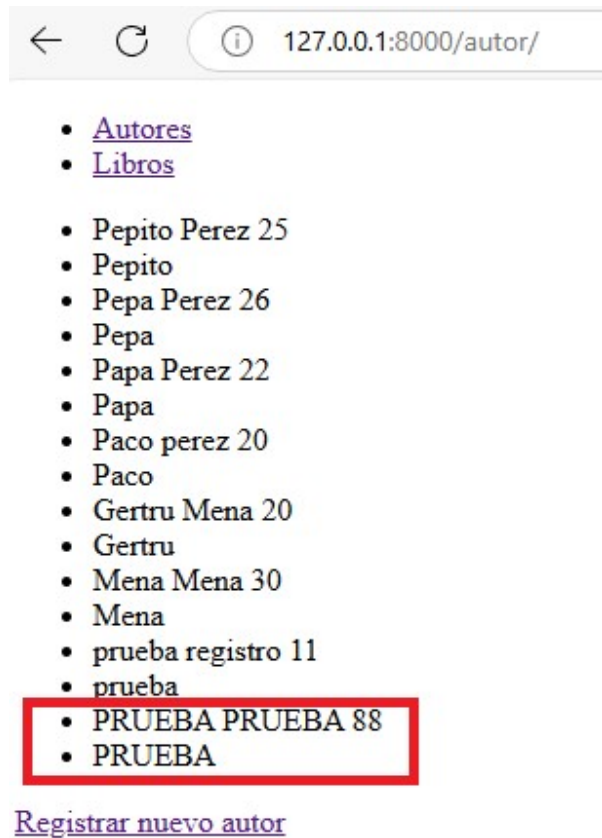
← ↻ ⓘ 127.0.0.1:8000/new-autor/

- [Autores](#)
- [Libros](#)

Nombre:

Apellido:

Edad:



## 2. Edición en Django

Para poder editar vamos a dar la opción en nuestra página web, por ejemplo seguimos con los autores. Añadimos el campo editar primero para que salga la opción de realizarlo.

```
{% extends 'index.html' %}
{% block title %} Listado de Autores {% endblock title %}

{% block content %}
<ul>
    {% for aut in authors %}
        <li>{{ aut }}</li>
        <li>{{ aut.name }}</li>
        <li><a href="#">Editar</a></li>
    {% endfor %}
</ul>
<div><a href="/new-autor/">Registrar nuevo autor</a></div>
{% endblock content %}
```

Igual que creamos un autor\_create en nuestra views.py, ahora hay que crear un autor\_update que recibirá como parámetro el id del autor que queremos modificar. Después de crear la función,

habrá que definir la url asociada a esa función. Inicialmente, el código de nuestra función `autor_update` sería como el siguiente que vamos a ir modificando posteriormente:

```
def autor_update (request, pk=None): #Recibe la clave del autor que queremos actualizar.
    autor=Author.objects.get(pk=pk) #Nos buscará el registro que coincida con la clave que le
    pongamos aquí. En el caso en el que no lo encuentre, va a generar un error, con lo que hay que
    capturar excepción
    #Existe otra opción que es con el filter, pero para grandes cantidades de datos el get es más
    óptimo
    # autor=Author.objects.filter(pk=pk).first Esta sería la otra forma
    return render (request,'update_autor.html',{'author':autor}) #si la página la tenemos dentro de
    template/core, hay que poner core/update_autor.html
```

Nuestra `urls.py` quedaría de la siguiente forma, pues como he comentado antes hay que añadir la nueva función:

```
from django.contrib import admin
from django.urls import path
#from . import views # Importo nuestro módulo views

from core.views import home,autor_list,libro_list,autor_create,autor_update

urlpatterns = [
    #path('hola/', views.index), #indicamos que la vista la queremos mostrar en esa ruta.
    path('', home), #quí no habría que meter la carpeta

    #path('otramas/', views.home),
    path('admin/', admin.site.urls),
    path('autor/',autor_list),
    path('libro/',libro_list),
    path ('new-autor/',autor_create),
    path ('update-autor/<int:pk>',autor_update) #Hay que pasarle el pk por parámetro. Para
    realizarlo se hace entre símbolos de <> indicando en primer lugar el tipo del campo, en este caso
    int y luego el nombre del campo pk.
]
```

Y nuestra `autor.html` hay que añadir la url adecuada, para que pueda enviar el identificador de autor:

```
{% extends 'index.html' %}
{%block title%} Listado de Autores {%endblock title%}

{%block content%}
<ul>
    {% for aut in authors %}
        <li>{{ aut }}</li>
        <li>{{ aut.name }}</li>      <!-- Lo que contiene la variable autor es una lista -->
        <li><a href="/update-autor/{{aut.id}}">Editar</a></li>
    {% endfor %}
</ul>
<div><a href="/new-autor/">Registrar nuevo autor</a></div>
{%endblock content%}
```



Hay que crear también la página `update_autor.html` que será similar a la de creación. En vez de crear los campos en html en los que podríamos introducir una modificación, vamos a utilizar también el `AuthoForm`, como hicimos con la creación. Para ello habrá que añadir en nuestra `views.py`, en la función en concreto de `update`, las siguientes líneas:

```
def autor_update (request, pk=None): #Recibe la clave del autor que queremos actualizar.
    autor=Author.objects.get(pk=pk) #Nos buscará el registro que coincida con la clave que le
    pongamos aquí.En el caso en el que no lo encuentre, va a generar un error, con lo que hay que
    capturar excepción
    #Existe otra opción que es con el filter, pero para grandes cantidades de datos el get es más
    óptimo
    # autor=Author.objects.filter(pk=pk).first Esta sería la otra forma

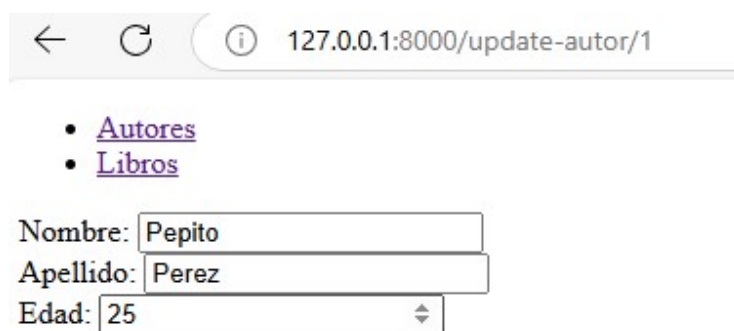
    author_form=AuthorForm(instance=autor) #todo form basado en un modelo tiene este atributo que hace
    referencia a la instancia de un objeto de la BD. Al pasarle este objeto, Django toma los atributos
    de éste y asociarlo a su campo html correspondiente que le estamos generando con el Form.
    return render (request,'update_autor.html',{'author':autor,'author_form':author_form}) #si la
    página la tenemos dentro de template/core, hay que poner core/update_autor.html
    #Le pasamos también la variable author_form para obtener los valores que deseamos editar.
```

Si recargo en mi navegador y con el siguiente código en nuestra página `update_autor.html`:

```
{% extends 'index.html' %}
{%block title%} Editar un autor {{author}} {%endblock title%} <!-- author es la variable que
recibe como parámetro -->

{%block content%}
{{author_form}}

{%endblock content%}
```



← ↻ ⓘ 127.0.0.1:8000/update-autor/1

- [Autores](#)
- [Libros](#)

Nombre:

Apellido:

Edad:

Igualmente aquí, hay que distinguir entre los dos tipos de información que tratamos, si es una solicitud, estamos mostrando, si estamos actualizando lo que realmente estamos recibiendo información. En resumen si es Get o Post, similar a la creación de autores. Nuestro `autor_update` hay que modificarlo para que recoja las dos situaciones

```
def autor_update (request, pk=None): #Recibe la clave del autor que queremos actualizar.

    autor=Author.objects.get(pk=pk) #Nos buscará el registro que coincida con la clave que le
    pongamos aquí.En el caso en el que no lo encuentre, va a generar un error, con lo que hay que
    capturar excepción
    #Existe otra opción que es con el filter, pero para grandes cantidades de datos el get es más
    óptimo
    # autor=Author.objects.filter(pk=pk).first Esta sería la otra forma
    if request.method == 'GET':

        autor_form=AuthorForm(instance=autor) #todo form basado en un modelo tiene este atributo que
        hace referencia a la instancia de un objeto de la BD. Al pasarle este objeto, Django toma los
        atributos de éste y asociarlo a su campo html correspondiente que le estamos generando con el Form.
        return render (request,'update_autor.html',{'author':autor,'author_form':autor_form}) #si la
        página la tenemos dentro de template/core, hay que poner core/update_autor.html
        #Le pasamos también la variable autor_form para obtener los valores que deseamos editar.

    if request.method == 'POST':
        autor_form=AuthorForm(data=request.POST, instance=autor) #Combinamos obtene información así
        como asociarla a una instancia. De esta forma Django entiende que es una actualización
        #El resto del proceso será igual que en la creación de autores.
        if autor_form.is_valid():
            autor_form.save()
            return redirect ('/autor/') #Se realiza igual que en la creación
        else: #Creo nueva instancia con la misma información que tenía y además pinto la información
        en el template
            autor_form=AuthorForm(data=request.POST, instance=autor)
            return render (request,'update_autor.html',{'author':autor,'author_form':autor_form})
```

Ahora tenemos que modificar nuestro update\_autor.html para que pinte correctamente la información.

```
{% extends 'index.html' %}
{%block title%} Editar un autor {{author}} {%endblock title%} <!-- author es la variable que
recibe como parámetro -->

{%block content%}
<form method="POST">
    {{author_form}}
    <button type="submit">Guardar</button>

</form>

{%endblock content%}
```

El resultado sería el siguiente:

← → ↻ ⓘ 127.0.0.1:8000/update-autor/1

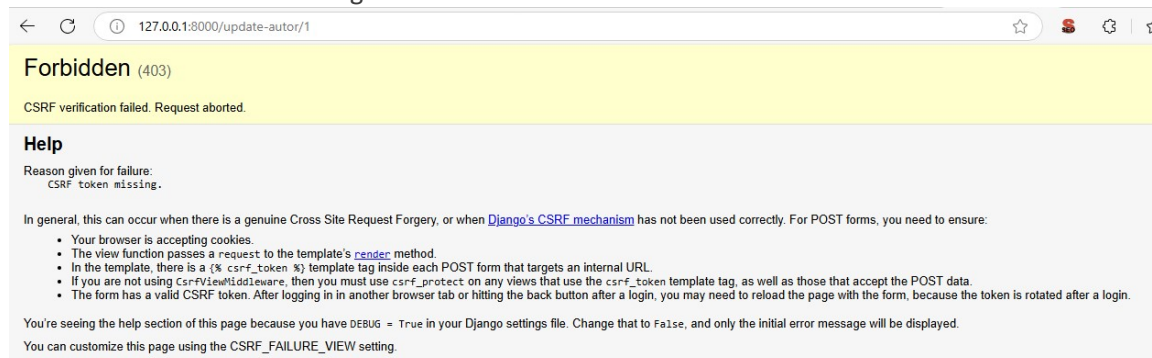
- [Autores](#)
- [Libros](#)

Nombre:

Apellido:

Edad:

Tal cual si le damos a guardar nos va a salir el siguiente error, fallo en la verificación del token que solucionamos añadiendo la siguiente línea:



```
{% extends 'index.html' %}
{%block title%} Editar un autor {{author}} {%endblock title%} <!-- author es la variable que
recibe como parámetro -->

{%block content%}
<form method="POST">
    {%csrf_token%}
    {{author_form}}
    <button type="submit">Guardar</button>

</form>
{%endblock content%}
```

Vuelvo a probar a editar mi autor pepito perez a cambiarle el apellido por ejemplo



← ↻ ⓘ 127.0.0.1:8000/update-autor/1

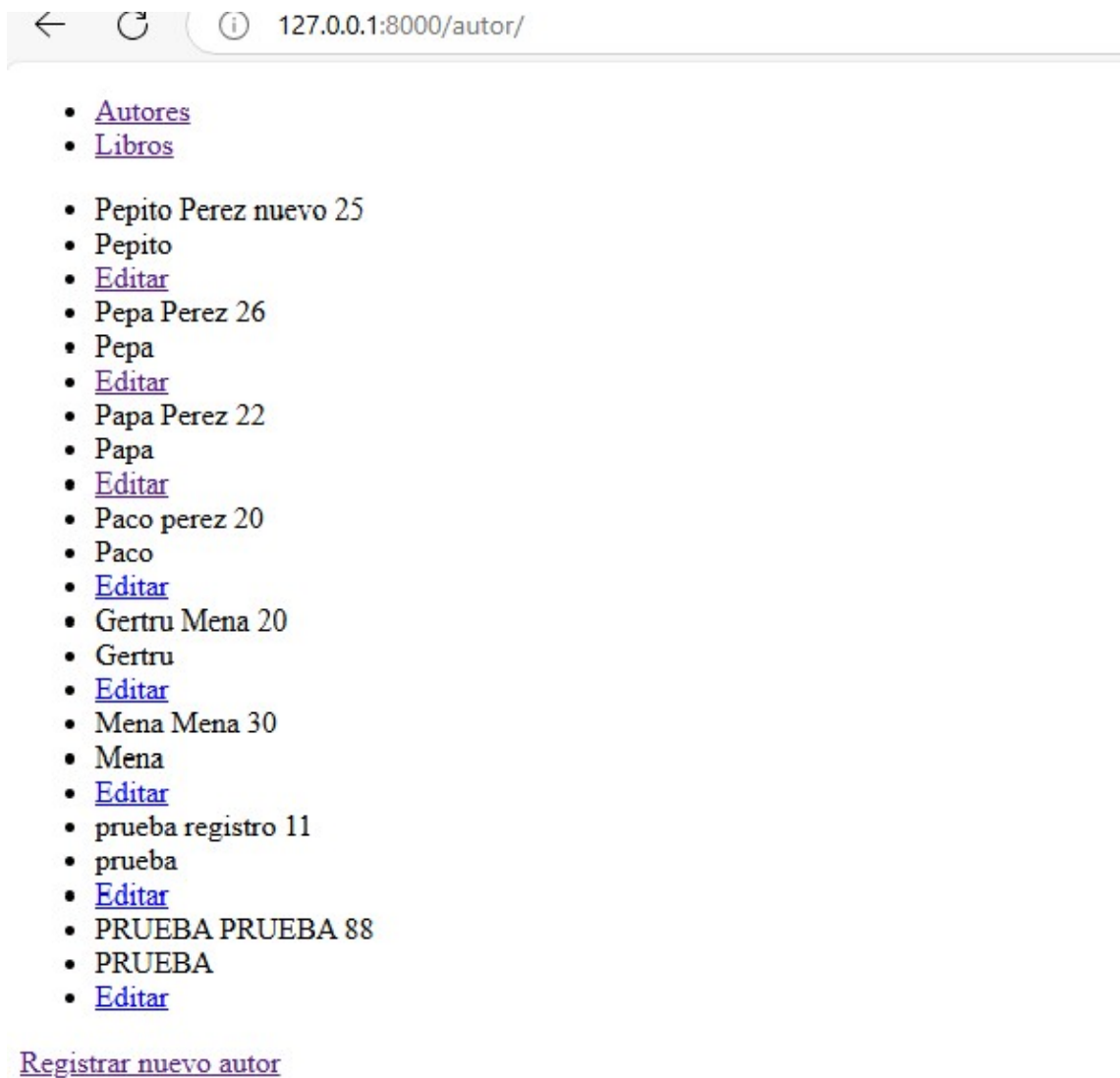
- [Autores](#)
- [Libros](#)

Nombre:

Apellido:

Edad:

Al darle a guardar, se va a la pagina de autores con la modificación realizada:



← ↻ ⓘ 127.0.0.1:8000/autor/

- [Autores](#)
- [Libros](#)

- Pepito Perez nuevo 25
- Pepito
- [Editar](#)
- Pepa Perez 26
- Pepa
- [Editar](#)
- Papa Perez 22
- Papa
- [Editar](#)
- Paco perez 20
- Paco
- [Editar](#)
- Gertru Mena 20
- Gertru
- [Editar](#)
- Mena Mena 30
- Mena
- [Editar](#)
- prueba registro 11
- prueba
- [Editar](#)
- PRUEBA PRUEBA 88
- PRUEBA
- [Editar](#)

[Registrar nuevo autor](#)

Si hacemos la prueba de meter espacios en blanco por ejemplo en el apellido va a dar el error, pues en el modelo hemos restringido a que no pueda ser en blanco.

### 3. Eliminación en Django

En primer lugar incluiremos la opción de eliminar en nuestra página de ejemplo, en este caso en autor.html

```
{% extends 'index.html' %}
{% block title%} Listado de Autores {%endblock title%}

{% block content%}
<ul>
    {% for aut in authors %}
        <li>{{ aut }}</li>
        <li>{{ aut.name }}</li>          <!-- Lo que contiene la variable autor es una lista -->
        <li><a href="/update-autor/{{aut.id}}">Editar</a></li>
        <li><a href="">Eliminar</a></li>
    {% endfor %}
</ul>
<div><a href="/new-autor/">Registrar nuevo autor</a></div>
{%endblock content%}
```



A nivel de vistas, hay que crear una nueva, autor\_delete. En esta función no vamos a tener un GET, pues no hay ninguna consulta, solamente necesitamos un POST. La función sería algo así:

```
def autor_delete (request, pk=None):

    #Eliminación directa
    Author.objects.filter(pk=pk).delete()
    #Otra forma de hacerlo
    #autor=Author.objects.get(pk=pk)
    #autor.delete()
    return redirect ('/autor/')
```

Tras crear la función vamos a dar de alta la url

```
urlpatterns = [
    #path('hola/', views.index), #indicamos que la vista la queremos mostrar en esa ruta.
    path('', home), #quí no habría que meter la carpeta

    #path('otramas/', views.home),
    path('admin/', admin.site.urls),
    path('autor/', autor_list),
    path('libro/', libro_list),
    path('new-autor/', autor_create),
    path('update-autor/<int:pk>', autor_update) #Hay que pasarle el pk por parámetro. Para
    realizarlo se hace entre símbolos de <> indicando en primer lugar el tipo del campo, en este caso
    int y luego el nombre del campo pk.
    path('delete-autor/<int:pk>', autor_delete)
]
```

Y la incluimos en nuestro href de la página:

```
{% extends 'index.html' %}
{%block title%} Listado de Autores {%endblock title%}

{%block content%}
<ul>
    {% for aut in authors %}
        <li>{{ aut }}</li>
        <li>{{ aut.name }}</li>          <!-- Lo que contiene la variable autor es una lista -->
        <li><a href="/update-autor/{{aut.id}}">Editar</a></li>
        <li><a href="/delete-autor/{{aut.id}}">Eliminar</a></li>
    {% endfor %}
</ul>
<div><a href="/new-autor/">Registrar nuevo autor</a></div>
{%endblock content%}
```

Después de incorporar estos cambios, la página ya borraría. Sería interesante que nuestra web diera un mensajito antes de borrar para realizar confirmación del borrado. ( A realizar por el alumno)