

1.Introducción

Tras ver cómo podemos crear modelos en BD, así como obtener información de esos modelos, a continuación veremos cómo trasladar al usuario que va a interactuar con nuestra página web, la vista de esos datos que hay en los modelos/tablas.

Cuando creamos una aplicación vamos a tener por defecto el fichero views.py en cada una de ellas. En este fichero se va a definir la lógica cómo tal, es decir, va a definir qué es lo que se hace cuando se ingresa a una url en especial.

Inicialmente vamos a escribir todas nuestras vistas en este fichero. Trae por defecto el import de render que nos permite enviar información para ser “pintada” en los navegadores. Se suele utilizar para enviar información a un template determinado y ésta se encarga de darle el formato y estilo adecuado.

Las vistas las podemos definir con funciones o con clases. Si recordamos nuestra views.py de la aplicación principal de nuestro proyecto, nuestras vistas las definimos con funciones (esto era lo que teníamos dentro de views.py):

```
def index (request): # El request captura las peticiones de los clientes
    return HttpResponse("<h1>hola mundo</h1>")
#Con esto hemos habilitado una vista, pero hay que de alguna manera darla de
alta, en urls.py

def home (request): # Pinta una página con render, también hay que darlo de
alta en urls.py
    return render(request, 'unodjango/index.html') # la página index.html hay
que crearla dentro del archivo de configuración de todo proyecto de django,
settings.py
```

Comenzaremos a trabajar con las vistas basadas en funciones para ir progresando poco a poco a las creadas con clases.

2.Vistas basadas en funciones

Hay que comentar, que toda función dentro de las vistas de Django va a tener un parámetro por defecto, request. No es más que una petición que hace el navegador o una entidad y es la información de quién nos está enviando la petición y qué nos está solicitando. (IP, usuarios que han iniciado sesión,etc).

Como ejemplo usaremos una función `home` que recibe `request` como parámetro y que va a devolver, usando `render`, algo que queremos pintar en la página web.

`Render` necesita varios parámetros para poder pintar. El primero de todos es un objeto `HttpRequest`, en este caso el parámetro `request`. Además necesita el nombre del template con el cual vamos a interactuar, en este caso vamos a llamarle `index.html`.

```
from django.shortcuts import render

# Create your views here.
def home (request):
    return render(request, 'index.html')
```

A continuación necesitamos crear una url que está asociada a esta vista/función. Para ellos nos iremos al fichero `url.py`, que se encuentra únicamente en la aplicación principal. Hay que saber que para crear una url dentro de Django se utiliza la función `path`, que viene importada por defecto y que toda url de Django tiene que ir dentro de la lista `urlpatterns`. Si no incluimos aquí la url, Django no la va a detectar

Hay que incluir nuestra función en el import

```
from core.views import home

urlpatterns = [
    path(
        path('', views.home), #Si no ponemos nada entre las comillas
        directamente #entra en nuestra página index.html
        path('admin/', admin.site.urls),
    ]
```

Hay que crear el fichero `index.html` en algún sitio, ¿dónde?. Si nos vamos al fichero `setting.py` dentro de nuestra aplicación principal, ahí vemos una parte que se llama `TEMPLATES` que es un diccionario. Ahí dentro existe una clave dentro de ese diccionario `DIRS`. Los valores de esta clave se introducen en una lista. En esa lista podemos introducir todas aquellas rutas dónde Django puede ir a buscar los templates. Normalmente se crea una carpeta 'template', al mismo nivel que las aplicaciones, dónde incluimos las páginas que vamos a visualizar. Se suele abrir dentro de template una carpeta por cada aplicación.

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': ['templates'], #creo directorio dentro de mi aplicación
        #dónde tendré mi index.html y así no dará error en el navegador.
        'APP_DIRS': True,
```

```

        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    ],
]

```

Aún así, la estructura de carpetas depende del grupo de trabajo que esté desarrollando. También es común, abrir una carpeta template dentro de cada aplicación y poner ahí las páginas. En este caso, habría que incluir la ruta de esta forma:

```
'DIRS': ['templates', 'core.template']
```

'APP_DIRS': `True`, con esta clave, decimos que vaya a buscar también en carpetas dentro de las aplicaciones.

3. Enviar Contexto a Templates

Una vez que ya tenemos nuestro modelo y sabemos utilizar vistas, vamos ahora a tratar de hacer visible los datos de nuestro modelo en esas vistas.

Vamos a usar las instrucciones de manejo del ORM de Django aprendidas usando el Shell, pero ahora las integraremos en nuestras views.

Para ellos nos vamos a nuestro archivo de la aplicación views.py y vamos a importar los modelos que utilizaremos. Comenzaremos con Author. Así mismo nos traemos los datos de autores que haya en la BD, usando una de las instrucciones del ORM de Django.

```

from django.shortcuts import render
#importamos los modelos a utilizar
from core.models import Author

def home (request):
    authors=Author.objects.all() #Aquí tendría a los autores pero hay que
    enviarlos al index.html
    return render(request, 'index.html')

```

Render, además de request y la plantilla, tiene un parámetro más, que es el contexto, es decir la información que uno quiere enviar hacia el template que le estamos indicando. Este contexto es de tipo diccionario.

```
def home (request):
    autor=Author.objects.all() #Aquí tendría a los autores pero hay que
    enviarlos al index.html

    return render(request, 'index.html',{'authors':autor}) #haciendo uso del
    diccionario, enviamos el contenido de la tabla author a la nuestro plantilla
    index.html
```

Hay que entender/aprender ahora, la sintaxis de Django, para que en nuestro index.html podamos mostrar el contenido de la tabla Authors. Usando un sistema de etiquetas procedemos a incorporar el código en nuestro index.html. Esto se llama el sistema de plantillas de Django.

Con las { dobles, mostramos el contenido de la variable authors, que es la clave del diccionario enviado por parámetro usando el render.

```
<h1>esto es el index</h1>
{{authors}}
```

Ahora aparecerá en el navegador la información de la tabla Author. Nos va a visualizar un QuerySet

Para visualizarlo de forma correcta, hay que realizar un tratamiento de la información así como el formateo correcto en nuestra página web, usando listas en Html y el bucle for usando el sistema de plantillas de Django.

```
<h1>esto es el index hola caracola,estoy</h1>
<ul>
    {% for aut in authors %}
        <li>{{ aut }}</li>
    {% endfor %}
</ul>
```

<!-- Lo que contiene la variable autor es una lista -->



Listado de autores

- Pepito Perez 25
- Pepa Perez 26
- Papa Perez 22
- Paco perez 20
- Gertru Mena 20
- Mena Mena 30

También podemos acceder a los campos de la tabla de la siguiente forma:

```
<h1>Listado de autores</h1>
<ul>
    {% for aut in authors %}
        <li>{{ aut }}</li>
        <li>{{ aut.name }}</li>
    {% endfor %}
</ul>
```

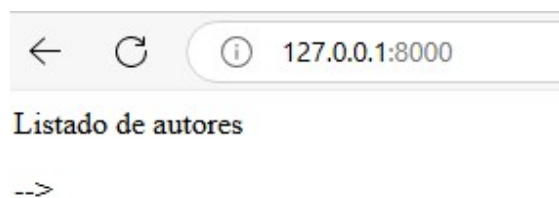
Si quisiéramos visualizar los libros, sabiendo que tenemos la relación [ManyToManyField](#).
Teniendo en mi view lo siguiente:

```
def home (request):
    #autor=Author.objects.all() #Aquí tendría a los autores pero hay que
    enviarlos al index.html
    book=Book.objects.all()
    #return render(request,'index.html',{'authors':autor}) #haciendo uso del
    diccionario, enviamos el contenido de la tabla author a la nuestro plantilla
    index.html
    return render(request,'index.html',{'books':book})
```

El index.html quedaría:

```
<h1>Listado de libros</h1>
<ul>
    {% for bo in books %}
        <li>{{ bo }}</li>
        <li>{{bo.author}}</li>    <!-- Lo que contiene la variable
autor es una lista -->
    {% endfor %}
</ul>
```

Y el resultado en el navegador:



Listado de libros

- 222 La gran princesa
- core.Author.None
- 636 El principito
- core.Author.None
- 98652 El gran misterio
- core.Author.None
- 968 La botella de vino
- core.Author.None
- 3444 El pastor alemán
- core.Author.None

No obtiene el valor requerido para el nombre del autor. Obtiene la referencia de la relación con el modelo Author, pero no el valor. Para poder obtenerlo , hay que modificar el código de esta manera:

```
<h1>Listado de libros</h1>
<ul>
    {% for bo in books %}
        <li>{{ bo }}</li>
        <li>{% for au in bo.author.all %}
            {{au}}
        {% endfor %}
        </li>
    {% endfor %}
</ul>
```

Listado de libros

- 222 La gran princesa
-
- 636 El principito
- Pepito Perez 25 Gertru Mena 20
- 98652 El gran misterio
-
- 968 La botella de vino
-
- 3444 El pastor alemán
- Pepito Perez 25 Pepa Perez 26

Aparece el autor que había en BD, por cada libro. Recordemos que con la relación `ManyToManyField`, un libro podía tener varios autores.

Si queremos formatear el contenido en una tabla:

Listado de libros

Titulo	Codigo	Autor
La gran princesa	222	
El principito	636	Pepito Perez 25 Gertru Mena 20
El gran misterio	98652	
La botella de vino	968	
El pastor alemán	3444	Pepito Perez 25 Pepa Perez 26

El código sería algo así:

```
<h1>Listado de libros</h1>
<table border="2">
  <thead>
    <th>Titulo</th>
    <th>Codigo</th>
    <th>Autor</th>
  </thead>
  <tbody>

    {% for bo in books %}
    <tr>
      <td>{{ bo.title }} </td>
      <td>{{ bo.cod }}</td>
      <td>{% for au in bo.author.all %}
        {{au}}
      {% endfor %}
      </td>
    </tr>
    {% endfor %}

  </tbody>
</table>
```

4.Sistema de plantillas de Django

El objetivo del sistema de plantillas de Django, es tratar de optimizar la reutilización de código. En este sistema de plantillas, se usan los tags y bloques, que lo que tratan es de agrupar código o estructuras html para su reutilización. Igual como en programación orientada a objeto heredamos los métodos de otras clases, aquí podemos agrupar bloques de código html que podremos utilizar en otras páginas y modificarlo según nos convenga. En general :

-Django permite la herencia de plantillas para que puedas reutilizar estructuras comunes en varias páginas.

-Plantilla Base: Contiene la estructura común y define bloques con {% block %} que las plantillas hijas pueden sobrescribir. Ejemplo:

```
<html>
  <head><title>{% block title %}Title{% endblock %}</title></head>
  <body>
```



```

    <header>{% block header %}{% endblock %}</header>
    <main>{% block content %}{% endblock %}</main>
</body>
</html>

```

En nuestro ejemplo, la idea sería renderizar desde nuestro archivo views.py , solamente el html, sin enviar ningún contexto, de la siguiente forma:

```

from django.shortcuts import render
#importamos los modelos a utilizar
from core.models import Author, Book

def home (request):
    #autor=Author.objects.all() #Aquí tendría a los autores pero hay que
    enviarlos al index.html
    #book=Book.objects.all()
    #return render(request, 'index.html', {'authors': autor}) #haciendo uso del
    diccionario, enviamos el contenido de la tabla author a la nuestro plantilla
    index.html
    #return render(request, 'index.html', {'books': book})
    return render(request, 'index.html')
#def home1 (request):
    # book=Book.objects.all()
    # return render(request, 'index.html', {'books': book})

```

Igualmente, en nuestro index.html ya no tendría sentido la estructura que teníamos. El contenido que había ahí, lo vamos a copiar y pegar en un nuevo archivo autor.html y libro.html, ambos archivos colgados en la misma ruta que el index.html.

Autor.html

```

<h1>Listado de autores</h1>
<ul>
    {% for aut in authors %}
        <li>{{ aut }}</li>
        <li>{{ aut.name }}</li>
    {% endfor %}
</ul>

```

es una lista --> <!-- Lo que contiene la variable autor

Libro.html

```
<h1>Listado de libros</h1>
<table border="2">
  <thead>
    <th>Titulo</th>
    <th>Codigo</th>
    <th>Autor</th>
  </thead>
  <tbody>

    {% for bo in books %}
    <tr>
      <td>{{ bo.title }} </td>
      <td>{{ bo.cod }}</td>
      <td>{% for au in bo.author.all %}
        {{au}}
      {% endfor %}
      </td>
    </tr>
    {% endfor %}
  </tbody>
</table>
```

Index.html

Ponemos html:5 y sale automáticamente el código html inicial. Añadimos los enlaces a las páginas correspondientes.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Blog de autores y libros</title>
</head>
<body>
  <ul>
    <li><a href="autor.html">Autores</a></li>
    <li><a href="libro.html">Libros</a></li>
  </ul>

</body>
</html>
```

Ahora realizamos los cambios en nuestra views.py para incorporar una nueva función con la llamada a autor.html y con la llamada a libro.html, de tal manera que nuestra view.py quedaría:

```
from django.shortcuts import render
#importamos los modelos a utilizar
from core.models import Author,Book

def home (request):
    #autor=Author.objects.all() #Aquí tendría a los autores pero hay que
    enviarlos al index.html
    #book=Book.objects.all()
    #return render(request,'index.html',{'authors':autor}) #haciendo uso del
    diccionario, enviamos el contenido de la tabla author a la nuestro plantilla
    index.html
    #return render(request,'index.html',{'books':book})
    return render(request,'index.html')
#def home1 (request):
    # book=Book.objects.all()
    # return render(request,'index.html',{'books':book})

    #Creamos una nueva función para enviar la información a nuestra página
    autor.html

def autor_list (request):
    autor=Author.objects.all()
    return render(request,'autor.html',{'authors':autor})

def libro_list (request):
    libro=Book.objects.all()
    return render(request,'libro.html',{'books':libro})
```

Así mismo, a estas nuevas funciones autor_list, y libro_list hay que asociarlo a una url, en el archivo urls.py

```
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

urlpatterns = [
    #path('hola/', views.index), #indicamos que la vista la queremos
    mostrar en esa ruta.
```

```
path('', home), #quí no habría que meter la carpeta

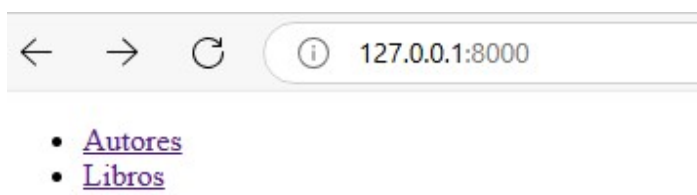
#path('otramas/', views.home),
path('admin/', admin.site.urls),
path('autor/', autor_list),
path('libro/', libro_list),
]
```

Así mismo en nuestro index.html, pondríamos los siguientes enlaces:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Blog de autores y libros</title>
</head>
<body>
  <ul>
    <li><a href="/autor/">Autores</a></li>
    <li><a href="/libro/">Libros</a></li>
  </ul>

</body>
</html>
```

Nuestro index se visualizará así :



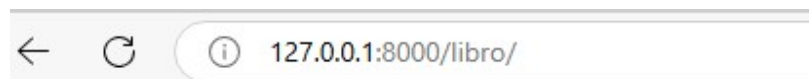
Autores mostrará lo siguiente:



Listado de autores

- Pepito Perez 25
- Pepito
- Pepa Perez 26
- Pepa
- Papa Perez 22
- Papa
- Paco perez 20
- Paco
- Gertru Mena 20
- Gertru
- Mena Mena 30
- Mena

Libros mostrará:



Listado de libros

Titulo	Codigo	Autor
La gran princesa	222	
El principito	636	Pepito Perez 25 Gertru Mena 20
El gran misterio	98652	
La botella de vino	968	
El pastor alemán	3444	Pepito Perez 25 Pepa Perez 26

¿Que está sucediendo?. Que nos está faltando en estas dos últimas páginas, listado de libros y listado de autores, la estructura para poder tener las opciones de menú que había en el index.html, es decir que si estoy en la pagina listado de libros, para poder acceder a listado de autores tengo que dar para atrás, volver al index y darle a autores. Lógicamente no es operativo. Una opción sería copiar el código que hay en index.html y ponerlo en ambas páginas en listado de libros y en listado de autores para que así nos aparezca el mismo menú que aparece en el index, en cada una de esas páginas. El problema de esto es que estaríamos duplicando código y tendríamos que ir copiando siempre la estructura del index en todas las subpáginas que vayamos creando. Ponemos como ejemplo, copiar el código del index.html en la página libro.html:

```
<h1>Listado de libros</h1>
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Blog de autores y libros</title>
</head>
<body>
  <ul>
    <li><a href="/autor/">Autores</a></li>
    <li><a href="/libro/">Libros</a></li>
  </ul>

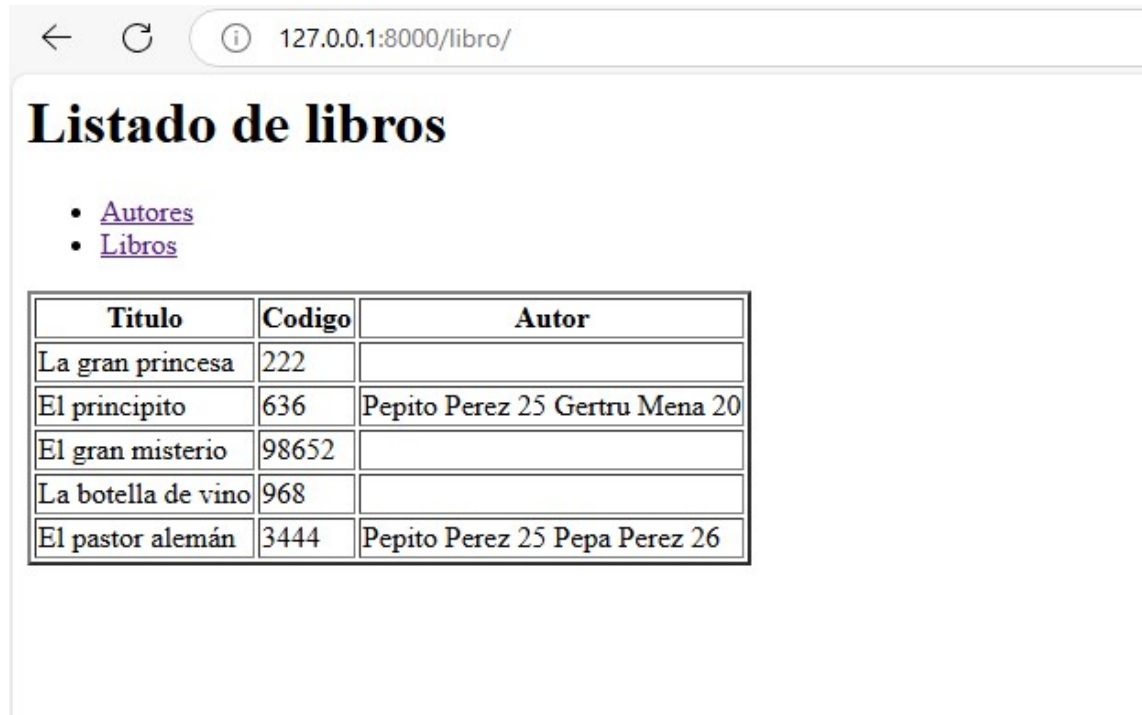
</body>
</html>

<table border="2">
  <thead>
    <th>Titulo</th>
    <th>Codigo</th>
    <th>Autor</th>
  </thead>
  <tbody>

    {% for bo in books %}
    <tr>
      <td>{{ bo.title }} </td>
      <td>{{ bo.cod }}</td>
      <td>{% for au in bo.author.all %}
        {{au}}
      {% endfor %}
      </td>
    </tr>
    {% endfor %}

  </tbody>
</table>
```

En resultado en el navegador sería:



Como puede apreciarse, ya aparece las dos opciones para movernos por el menú.

El problema, como he comentado antes, que tenemos que andar copiando el código del index.html en cada una de las subpáginas que vayamos creando. En esta caso sólo tenemos dos páginas, pero qué pasaría si tuviéramos bastantes más.

Aquí es donde entra la magia del sistema de plantillas de Django.

6.Introduciéndonos en el sistema de plantillas de Django

En Django, tenemos la posibilidad de incluir tags para poder incluir dentro del código html código en python, tal como un bucle for o para pintar variables, usando {% %}.

La idea es agrupar en bloques (que usan {% %} para su marcado) aquel código que queremos reutilizar y pueda ser susceptible de ser modificado según nos convenga.

Entonces en el código que queremos reutilizar vamos marcando los bloques. En el index, nos gustaría que el título por ejemplo, fuera variando en función de la página en la que esté. Es decir, que si entro en autores nos pusiera listado de autores en el título y si entro en libros nos salga listado de libros.

Vamos a incluir un bloque en el título de index.html

```

<!DOCTYPE html>

<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{%block title%} Blog de autores y libros {%endblock
title%}</title>
</head>
<body>
    <ul>
        <li><a href="/autor/">Autores</a></li>
        <li><a href="/libro/">Libros</a></li>
    </ul>

</body>
</html>

```

Estos tags de bloques, son invisibles a nivel de navegador. El bloque también lo podíamos haber puesto incluyendo más código, tal que así y va a funcionar igualmente:

```

{%block title%}
<title> Blog de autores y libros </title>
{%endblock title%}

```

Vamos a agregar también un bloque para el css, otro bloque del menú y otro bloque para el contenido.

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{%block title%} Blog de autores y libros {%endblock
title%}</title>
    {%block css%}
    {%endblock css%}
</head>
<body>
    {%block menu%}
    <ul>
        <li><a href="/autor/">Autores</a></li>
        <li><a href="/libro/">Libros</a></li>
    </ul>

```



```

    {%endblock menu%}
    {%block content%}
    {%endblock content%}
</body>
</html>

```

Estos bloques podrán ser reescritos en las plantillas que hereden de index.html.

Para indicar esto último, me iría a las páginas autor.html y libro.html y pondríamos el siguiente código al inicio para indicar que extienden de index.html. Hay que hacerlo en todas las páginas que queremos que reutilicen los bloques definidos en el index.html

```

{% extends 'index.html' %}
<h1>Listado de autores</h1>
<ul>
    {% for aut in authors %}
        <li>{{ aut }}</li>
        <li>{{ aut.name}}</li>      <!-- Lo que contiene la variable autor
es una lista -->
    {% endfor %}
</ul>

```

Al poner el extends y entrar en el listado de autores, tal como lo tenemos, no sale el listado de autores si no que sale directamente el mismo contenido que el de index.html



Esto ocurre porque al habérsele aplicado la herencia de plantillas sólo va a existir lo que dice index.html y todo lo que esté dentro de los bloques que éste contenga. En Autor.html, todo el código que hay debajo del extends, no está agrupado en ningún bloque, con lo que Django lo ignora.

Entonces en nuestra página autor.html, vamos a ir rellenando los bloques con el contenido que nos interese, pues todo lo que no esté dentro de un bloque no se va a mostrar.

```
{% 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>
{%endblock content%}
```



- [Autores](#)
- [Libros](#)
- Pepito Perez 25
- Pepito
- Pepa Perez 26
- Pepa
- Papa Perez 22
- Papa
- Paco perez 20
- Paco
- Gertru Mena 20
- Gertru
- Mena Mena 30
- Mena

Para el caso de la página libro.html, el código sería:

```
{% extends 'index.html' %}
<!--<h1>Listado de libros</h1>-->
{%block title%} Listado de Libros {%endblock title%}
{%block content%}
<table border="2">
    <thead>
        <th>Titulo</th>
        <th>Codigo</th>
        <th>Autor</th>
```

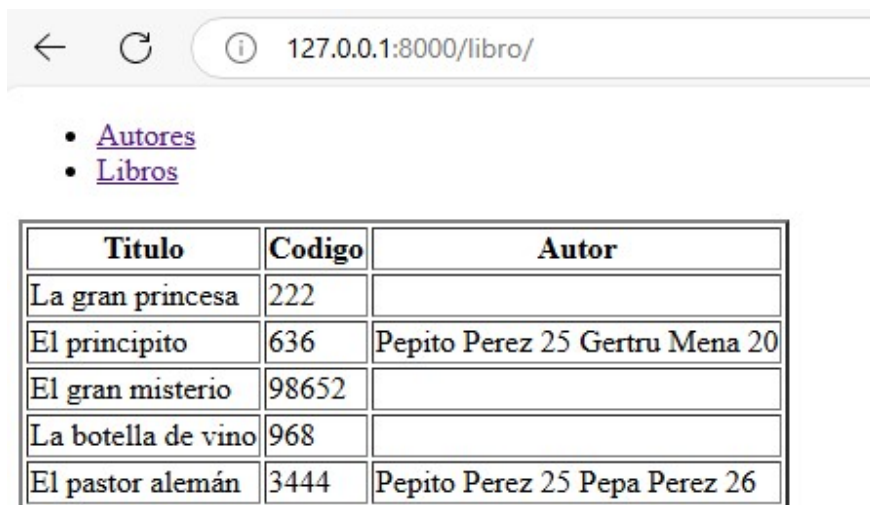
```

</thead>
<tbody>

    {% for bo in books %}
    <tr>
        <td>{{ bo.title }} </td>
        <td>{{ bo.cod }}</td>
        <td>{% for au in bo.author.all %}
            {{au}}
        {% endfor %}
        </td>
    </tr>
    {% endfor %}

</tbody>
</table>
{%endblock content%}

```



Titulo	Codigo	Autor
La gran princesa	222	
El principito	636	Pepito Perez 25 Gertru Mena 20
El gran misterio	98652	
La botella de vino	968	
El pastor alemán	3444	Pepito Perez 25 Pepa Perez 26

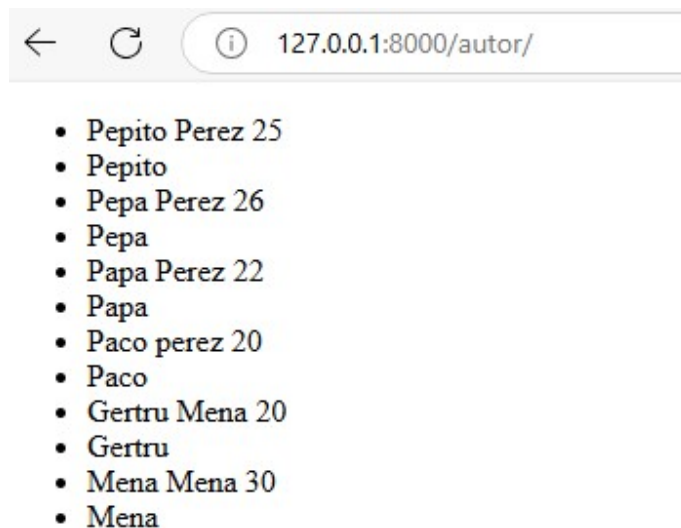
Vemos que no ha hecho falta escribir el código para que salga el menú en cada una de las subpáginas, pero aún así sale, pues heredan de index.html.

En conclusión, todo lo que no esté entre bloques en las subpáginas, no lo va a pintar. Así mismo, todo lo que queramos modificar en las subpáginas en relación a algún bloque que haya sido definido en el index.html, pues simplemente hay que poner las etiquetas de los bloques y especificar lo que queremos modificar, es decir, redefinir.

Por ejemplo, si en nuestro código de la página autor.html, ponemos el bloque menú y no incluimos nada, el menú no aparecerá.

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

El resultado de esto sería, lo mismo que antes pero sin menú.



En el index.html, es interesante añadir los bloques footer y javascript, para cuando agreguemos ese tipo de código.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>{%block title%} Blog de autores y libros {%endblock
title%}</title>
  {%block css%}
  {%endblock css%}
</head>
<body>
  {%block menu%}
  <ul>
    <li><a href="/autor/">Autores</a></li>
    <li><a href="/libro/">Libros</a></li>
  </ul>
  {%endblock menu%}
  {%block content%}
  {%endblock content%}
  {%block footer%}
  {%endblock footer%}
  {%block js%}
  {%endblock js%}
</body>
</html>
```

En general, vamos a tener visualmente lo mismo, pero con el sistema de plantillas de Django el código va a estar mucho más estructurado.