TEMA IV

DESARROLLO DE APLICACIONES WEB EN BACKEND

"Siempre parece imposible,
hasta que se hace"
Nelson Mandela

FUNDAMENTOS DE LA PROGRAMACIÓN WEB

7.2 INTRODUCCIÓN A LOS TEMPLATES

Por razones obvias no se puede trabajar como lo hemos hecho en el primer ejemplo, mezclando html y los datos para formar una cadena de texto y retornarla en la vista.

Debemos separar el diseño de la lógica de la aplicación, para ello tenemos los templates.

Una plantilla de Django es un texto que pretende separar la presentación de un documento de sus datos. Normalmente, las plantillas son usadas para producir HTML, pero las plantillas de Django son igualmente capaces de generar cualquier formato basado en texto.

Vamos a empezar con los conceptos más básicos e iremos añadiendo variables, etiquetas y filtros.

Para utilizar un template:

1. Crear una carpeta llamada templates y colocar dentro el html correspondiente a la plantilla

2. Añadir esta ruta en settings

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        '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',
            ],
        },
    },
]
```

3. Crear un contexto(diccionario): datos y funciones que queremos que pasarle al template

4. Renderizar el Template

render(request, plantilla)

render(request, plantilla, datos)

Ejemplo: "Hola Mundo con plantilla"

PASAR DATOS A LAS PLANTILLAS DESDE LA VISTA

Para poder utilizar variables de nuestra vista en la plantilla, utilizaremos el contexto. Un contexto acepta como parámetro un diccionario, a través de esta estructura de python podemos pasar variables de cualquier tipo (incluidos objetos) a un template.

1. Definimos el diccionario con las parejas clave-valor y lo pasamos como parámetro del método render
2. Para utilizar dichos valores, en la plantilla {{clave}}

render(request, plantilla,diccionario)

Ejemplo2 : Variar el ejemplo anterior para saludar o despedirnos

Ejemplo3: Al ejemplo anterior le añadimos una variable con el nombre y le pasamos a la plantilla el saludo o despedida y dicho nombre.

CÓDIGO EN LAS PLANTILLAS

En las plantillas además de variables podemos incluir código python, es frecuente necesitar condicionales o bucles.

1. Cualquier texto entre un par de llaves es una variable. Esto significa "insertar el valor de la variable a la que se dio ese nombre".
2. Cualquier texto que esté rodeado por llaves y signos de porcentaje (por ej. {% if ordered_warranty %}) es una etiqueta de plantilla. La definición de etiqueta es bastante amplia: una etiqueta sólo le indica al sistema de plantilla "haz algo".

Etiquetas

1. **Sentencias condicionales:** if, if – else, ifequal , ifnotequal

La sintaxis del if

```
{% if variable %}
      -----
{% else %}
      -----
{% end if %}
```

Esta condición mira si la variable existe y tiene valor, las dos cosas, por ejemplo en el caso de una lista mira si además de existir no está vacía. En el caso de cualquier variable mira si se la hemos pasado en el contexto. Esto es útil cuándo tenemos un template que puede recibir o no algún dato.

La parte else no es obligatoria , se pueden anidar.

Se pueden usar los operadores de comparación (==,!=,<=…) pero también podemos usar un par de etiquetas {% ifequal %} , {% ifnotequal %}

```
{% ifequal section 'sitenews' %}
    <h1>Site News</h1>

{% endifequal %}
```

Ejemplo: Variar el anterior para pasarle o no el nombre y controlarlo desde el template.

2. Bucles

La sintaxis del bucle for

```
{% for item in lista%}
        {{item}}
{% endfor %}
```

No se admite la "ruptura" de un bucle antes de que termine (break). De manera similar, no se soporta la sentencia continue que se encarga de retornar inmediatamente al inicio del bucle.

La etiqueta {% for %} asigna la variable forloop a la plantilla con el bucle. Esta variable tiene algunos atributos con información sobre del progreso del bucle:

- forloop.counter: número entero que almacena el número de veces que se ha entrado en el bucle. Esta es indexada a partir de 1.

```
{% for item in todo_list %}
    <p>{{ forloop.counter }}: {{ item }}</p>
{% endfor %}
```

- forloop.counter0 es como forloop.counter, excepto que esta es indexada a partir de cero.
- forloop.revcounter almacena en todo momento el número de iteraciones que faltan para terminar el bucle. La primera vez que se ejecuta el bucle for será igual al número de elementos que hay en la secuencia.
- forloop.first su valor es True si es la primera vez que se pasa por el bucle. Esto es conveniente para ocasiones especiales:

```
{% for object in objects %}
    {% if forloop.first %}<li class="first">{% else %}<li>{% endif %}
        {{ object }}
    </li>
{% endfor %}
```

- forloop.last True si es la última pasada por el bucle. Un uso común para esto es poner un carácter pipe entre una lista de enlaces:

```
{% for link in links %}{{ link }}{% if not forloop.last %} | {% endif %}{% endfor %}
```

El código de la plantilla de arriba puede mostrar algo parecido a esto:

Link1 | Link2 | Link3 | Link4

- forloop.parentloop esta es una referencia al objeto padre de forloop, en el caso de bucles anidados. Aquí un ejemplo::

```
{% for country in countries %}
   <table>
   {% for city in country.city_list %}
      <tr>
```

```
    <td>Country #{{ forloop.parentloop.counter }}</td>
    <td>City #{{ forloop.counter }}</td>
    <td>{{ city }}</td>
    </tr>
  {% endfor %}

  </table>{% endfor %}
```

La variable mágica foorloop está sólo disponible dentro de bucles. Una vez terminado el bucle desaparece.

Si el bucle lo aplicamos a un diccionario podemos recorrer claves y valores

```
{% for key, value in data.items %}

  {{ key }}: {{ value }}

{% endfor %}
```

Contexto

Hemos comentado que a la plantilla podemos pasarle objetos, hay que tener en cuenta que las llamadas a los atributos del objeto y a sus métodos en las plantillas se hacen exactamente igual **cuándo los métodos no llevan parámetros**. Es decir no pondremos paréntesis.

Es más, en los templates utilizaremos el operador **.** Para llamar a:

- Clave del diccionario: cuando el contexto a su vez pasa un diccionario

```
person = {'name': 'Sally', 'age': '43'}
c = {'person': person}
En el Template
'{{ person.name }} is {{ person.age }} years old.'
```

- Atributo de un objeto
- Método de la clase

- Índice(posición) de una lista

```
c = {'items': ['apples', 'bananas', 'carrots']}
En el template
{{ items.2 }}
```

En este orden.

Los puntos pueden ser anidados

```
persona.nombre.upper
```

Ejemplo: Definimos una lista en una vista con los lenguajes de programación que sabemos. Pasamos esa lista a una plantilla que la recorre y muestra los lenguajes en una

Comentarios

Se pueden poner comentarios en las plantillas:

De una línea: {# texto #}

De varias líneas {% comment %} texto

```
texto
{% endcomment %}
```

Filtros (Pipes)

Un filtro o pipe es una herramienta que nos permite "transformar" visualmente la información, es decir presentarla con un formato

Por ejemplo, si escribo en un template

- {{nombre.upper}} estoy llamando al método upper de la clase String
- {{nombre | upper}} estoy llamando al filtro upper

Los filtros se pueden concatenar

{{nombre| first | lower}} primera letra de nombre en minúsculas

{{texto|escape| linebreaks}}convierte los saltos de línea en etiqueta p

Los filtros pueden llevar argumentos, van entre comillas y después de :

{{frase|truncatewords : "5"}} las 5 primeras palabras de frase

{{fecha|date:"d de F de Y"}} 4 de Enero de 2020

El filtro escape es muy útil porque nos permite quitar algunos caracteres (&, ", `…) de los datos proporcionados por el usuario

Otro ejemplo que nos puede resultar útil es length nos da la longitud de cualquier objeto python al que yo le pueda aplicar su función len

Hay muchos filtros, se pueden consultar en la página de django.

Modelos

Podemos pasar modelos a las vistas. Creamos una carpeta llamada models y ahí colocamos nuestras clases(el código de las mismas tal y como hemos visto).

Generamos el diccionario con la pareja clave-valor siendo el valor un objeto de la clase.