TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE

- Despliegue de páginas con contenido dinámico.
- Sirviendo contenido estático.
- Django y la configuración de archivos estáticos.
- Agregando Bootstrap a Django.
- Links y navegación.
- El modelo MVC en Django.
- Mostrar contenido estático con Bootstrap.
- Renderización dinámica y templates.
- ¿Cómo usar páginas parciales?
- Vistas en Django.
- Herencia de Plantillas.

DESPLIEGUE DE PÁGINAS CON CONTENIDO DINÁMICO

SIRVIENDO CONTENIDO ESTÁTICO

Las páginas estáticas contienen información que no cambia hasta que el diseñador o programador la modifica manualmente. Las páginas web estáticas, compuestas únicamente de HTML y CSS, se implementan fácilmente. Pero una de sus grandes limitaciones es el esfuerzo que se requiere para actualizarlas. Cambiar un solo elemento en una página web estática requiere reconstruir y recargar en el servidor toda la página, o a veces incluso un grupo de páginas web.

Este proceso es demasiado engorroso para una organización que con frecuencia necesita publicar información en tiempo real, tal como eventos. Además, durante este proceso, un desarrollador puede cambiar accidentalmente otros artículos en la página, arruinando seriamente la información de la web, o incluso el diseño completo del sitio.

Las páginas web dinámicas permiten cambiar fácilmente su contenido en tiempo real sin siquiera tocar el código de la página. Sin hacer manualmente cualquier cambio en la página en sí, su información puede variar. Esto hace posible mantener el contenido de la página actualizado para que lo que un visitante ve allí pueda ser actualizado o substituido en un día, una hora, o un minuto. El diseño central de la página web puede seguir siendo el mismo, pero los datos presentan cambios constantes.

Para crear con éxito una página web dinámica, se debe conocer un método para insertar automáticamente datos en tiempo real en el código HTML que se envía al navegador del cliente. Aquí es donde entran en juego los lenguajes de script, que permiten insertar código de programa dentro de una web, y genera dinámicamente HTML en el navegador del cliente.

DJANGO Y LA CONFIGURACIÓN DE ARCHIVOS ESTÁTICOS

La puesta en producción de archivos estáticos consta de dos pasos: ejecute el comando `collectstatic`, cuando cambien los archivos estáticos, y luego organice el directorio de archivos estáticos recopilados (`STATIC_ROOT`) para moverlos al servidor de archivos estáticos y servirlos. Dependiendo de `STATICFILES_STORAGE`, es posible que sea necesario mover los archivos a una nueva ubicación manualmente, o también el método `post_process` de la clase Storage podría ocuparse de eso.

DESPLEGAR UNA PÁGINA WEB ESTÁTICA

Django mediante: `django.contrib.staticfiles`, gestiona el contenido estático para las aplicaciones, y los ordena en una sola ubicación fácil de referenciar y de usar. El primer lugar donde inicia el manejo de los archivos estáticos reside en el archivo de configuraciones del proyecto: settings.py. En este archivo tenemos líneas exclusivamente dedicadas al manejo del contenido estático. Existen 4 elementos:

- `STATIC_ROOT`.
- `STATIC_URL`.
- `STATICFILES_DIRS`.
- `STATICFILES_FINDERS`.

Para mayor referencia y documentación, podemos consultar en: https://runebook.dev/es/docs/django/ref/settings#static-files

Cada uno de ellos con un propósito documentado en el mismo archivo `settings.py` a modo de comentario.

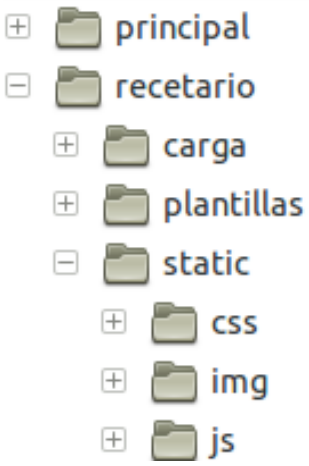
Debemos prestar atención a `STATICFILES_DIRS`. Este elemento permite declarar la ruta, desde la cual se enlazará el contenido estático, lo dejamos de la siguiente manera:

```
STATICFILES_DIRS = (
    # Put strings here, like "/home/html/static" or
"C:/www/django/static".
    # Always use forward slashes, even on Windows.
    # Don't forget to use absolute paths, not relative paths.
    os.path.join(RUTA_PROYECTO,'static'),
)
```

El resto de los elementos referentes al contenido estático no se manipulan, pues estamos en la versión de desarrollo y aún falta para la etapa de producción. No olvidar guardar el archivo, para proseguir sin errores.

DIRECTORIO STATIC

Ahora procedemos a crear el directorio: `static`. Este se debe hacer dentro del directorio del proyecto, al mismo nivel que: carga y plantillas. Dentro del directorio `static`, debemos tener una carpeta por cada tipo de contenido estático que deseemos incluir. Tendremos tres subdirectorios: `css, img y js`.



Dentro de cada uno de estos directorios, debemos incluir nuestro contenido estático. Se usarán instrucciones muy básicas tanto para CSS, como para JavaScript.

USO DE LIBRERÍAS PARA FRONTEND

Se pueden usar diversas herramientas existentes como: Bootstrap, Boilerplate, 960gs, Blueprint, jQuery Mobile, o cualquier otro conjunto de archivos que faciliten la construcción de interfaces de usuario.

AGREGANDO BOOTSTRAP A DJANGO

Cuando se está trabajando en las vistas, o en la parte Frontend del proyecto, una de las mejores opciones es Bootstrap. Antes de continuar, es importante que tengas desplegado Django y su gestor de paquetes PIP (Package Installer Python)

```
$ pip install django-bootstrap4
Collecting django-bootstrap4
  Downloading
https://files.pythonhosted.org/packages/80/ad/23a156a282c733c33bc728257
2b89d00131130519e0de18013e7e3fcb6e3/django-bootstrap4-0.0.6.tar.gz
Building wheels for collected packages: django-bootstrap4
  Running setup.py bdist_wheel for django-bootstrap4: started
  Running setup.py bdist_wheel for django-bootstrap4: finished with
status 'done'
  Stored in directory:
C:\Users\FRONTEND\AppData\Local\pip\Cache\wheels\a2\92\2f\110d092bcbf7e
25347880d2975e8c53bf1c33177ea3b360478
Successfully built django-bootstrap4
Installing collected packages: django-bootstrap4
Successfully installed django-bootstrap4-0.0.6
```

Ahora ve a tu archivo `settings.py`, y agrega Bootstrap en la parte `INSTALLED_APPS`.

```
# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'bootstrap4',
]
```

Luego, y de nuevo en el archivo `settings.py`, se agrega la ruta del directorio de los templates:

```
TEMPLATES = [
```

```
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],  # Acá
        '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',
            ],
        },
    },
]
```

En el archivo `urls.py` se agrega la ruta para visualizar una determinada vista que tenga Bootstrap.

```
urlpatterns = [
    path('', views.index, name='index'),  # Acá
    path('admin/', admin.site.urls),
]
```

En el archivo **`views.py`** se agrega la vista a la página index.html, que es la vista **`HTML`** en donde se muestran los elementos y la página en Bootstrap.

VIEWS.PY

```
from django.shortcuts import render_to_response

def index (request):
return render_to_response('index.html')
```

Se crea la carpeta llamada templates, y dentro de ella creamos un archivo llamado index.html. Lo que haremos es que entre las etiquetas **`<head></head>`**, se instancia las librerías CSS de Bootstrap.

INDEX.HTML

```
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">
    <meta name="description" content="">
    <meta name="author" content="">
    <link rel="icon" href="favicon.ico">

```

```
    <title>Como integrar Django y Bootstrap 4 </title>

          {# Cargamos la librería #}
          {% load bootstrap4 %}

          {# CSS Bootstrap #}
          {% bootstrap_css %}


  </head>
```

Para la librería jQuery y los archivos `JavaScript` de Bootstrap, antes de cerrar la etiqueta `</body>` debajo del documento `HTML`, se instancian.

```
          {# JS y jQuery Bootstrap #}
          {% bootstrap_javascript jquery='full' %}

  </body>
</html>
```

Finalmente, se puede agregar todo tipo de componentes de Bootstrap que necesitemos usar, todo los colocamos dentro de las etiquetas `<body></body>`.

AGREGANDO UN ELEMENTO JUMBOTRON DE BOOTSTRAP

Jumbotron es un componente Bootstrap que ocupa todo el ancho del puerto de visualización del navegador, y se utiliza para crear fácilmente secciones destacadas de ancho completo. Es como un cuadro de llamada a la acción para mostrar contenido importante y atraer usuarios.

El código para el jumbotron de ancho completo se proporciona a continuación. Como puede ver el código, la clase `".jumbotron-fluid"` se usa para el ancho completo junto con `".container"`.

```
<div class="jumbotron">

  <h1 class="display-4">Hello, world!</h1>

  <p class="lead">This is a simple hero unit, a simple jumbotron-style
component for calling extra attention to featured content or
information.</p>

```

```
  <hr class="my-4">

  <p>It uses utility classes for typography and spacing to space
content out within the larger container.</p>

  <a class="btn btn-primary btn-lg" href="#" role="button">Learn
more</a>

</div>
```

El código jumbotron puede usar enlaces CDN pre compilados Bootstrap CSS y archivos JS. Pero si desea alojar los archivos en su servidor, descargue y cargue los archivos requeridos desde el sitio oficial.

AGREGANDO UN ELEMENTO FOOTER DE BOOTSTRAP

Se utiliza para crear pie de páginas, y se inserta de la siguiente manera:

```
<div class="card text-center">
  <div class="card-header">
    Featured
  </div>
  <div class="card-body">
    <h5 class="card-title">Special title treatment</h5>
    <p class="card-text">With supporting text below as a natural lead-
in to additional content.</p>
    <a href="#" class="btn btn-primary">Go somewhere</a>
  </div>
  <div class="card-footer text-muted">
    2 days ago
  </div>
</div>
```

PÁGINAS RESPONSIVAS

Es un tipo de diseño utilizado en sitios web. Su función es adaptar contenido de múltiples formatos para dispositivos móviles. Los sitios web responsive cambian para ofrecer la mejor experiencia a los visitantes independientemente del dispositivo: teléfonos inteligentes, tabletas o computadoras de escritorio.

LINKS Y NAVEGACIÓN

Un link se reconoce por el hecho de que la flecha que indica la posición del mouse cambia en una mano derecha con el dedo de dirección extendido.

El elemento HTML `<nav>` representa una sección de una página cuyo propósito es proporcionar enlaces de navegación, ya sea dentro del documento actual, o a otros documentos. Ejemplos comunes de secciones de navegación son: menús, tablas de contenido e índices.

EL MODELO MVC EN DJANGO

El diseño Modelo-Vista-Controlador (`MVC`) define una forma de desarrollar software en la que el código para definir y acceder a los datos (el modelo) está separado del pedido lógico de asignación de ruta (el controlador), y que a su vez, está separado de la interfaz del usuario (la vista).

MVC VS. MTV

Modelo Vista Controlador MVC	Modelo Vista Plantilla MVT
MVC tiene un controlador que controla tanto el modelo como la vista.	MVT tiene vistas para recibir solicitudes HTTP y devolver respuestas HTTP.
Ver indica cómo se presentarán los datos del usuario.	Las plantillas se utilizan en MVT para ese propósito.
En MVC, tenemos que escribir todo el código específico del control.	La parte del controlador es administrada por el propio Framework.
Altamente acoplado.	Débilmente acoplado.
Las modificaciones son difíciles.	Las modificaciones son fáciles.
Adecuado para el desarrollo de aplicaciones grandes, pero no para aplicaciones pequeñas.	Adecuado tanto para aplicaciones pequeñas como grandes.
El flujo está claramente definido, por lo que es fácil de entender.	El flujo a veces es más difícil de entender en comparación con MVC.

No implica el mapeo de URL.	Se lleva a cabo la asignación de patrones de URL.
Los ejemplos son: ASP.NET MVC, Spring MVC, entre otros.	Django usa el patrón MVT.

MOSTRAR CONTENIDO ESTÁTICO CON BOOTSTRAP

A continuación, se muestra un ejemplo modal estático (lo que significa que su position y `display` se han anulado). Se incluyen el encabezado modal, el cuerpo modal (requerido para el `padding`) y el pie de página modal (opcional). Le pedimos que incluya encabezados modales con acciones de descarte siempre que sea posible, o proporcione otra acción de descarte explícita.

```
<div class="modal" tabindex="-1">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title">Modal title</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal"
aria-label="Close"></button>
      </div>
      <div class="modal-body">
        <p>Modal body text goes here.</p>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-secondary" data-bs-
dismiss="modal">Close</button>
        <button type="button" class="btn btn-primary">Save
changes</button>
      </div>
    </div>
  </div>
</div>
```

RENDERIZACIÓN DINÁMICA Y TEMPLATES

La función `render`, tal y como su nombre nos indica, nos permite renderizar un template. Esta función es de gran utilidad cuando deseamos responder a la petición de un cliente mediante alguna página web, o también podemos utilizarla, por ejemplo, para generar nuestros propios correos electrónicos.

Una vez que tienes un objeto Template, le puedes pasar datos brindando un contexto. Un contexto es simplemente un conjunto de variables y sus valores asociados. Una plantilla usa estas variables para llenar y evaluar estas etiquetas de bloque.

Un contexto es representado en Django por la clase `Context`. Ésta se encuentra en el módulo `django.template`. Su constructor toma un argumento opcional: un diccionario que mapea nombres de variables con valores. Llama al método `render()` del objeto Template con el contexto para "llenar" la plantilla.

```
1 from django.shortcuts import render
2
3 def my_view(request):
4     # View code here...
5     return render(request, 'myapp/index.html', {
6         'foo': 'bar',
7     }, content_type='application
```

Esta función recibe como argumento, de forma obligatoria, 3 valores. La petición a perse, el template a renderizar, y un contexto. El contexto no será más que un diccionario cuyas llaves podrán ser utilizadas dentro del template. Mas información la podemos encontrar en: https://docs.djangoproject.com/en/4.0/topics/http/shortcuts/

¿CÓMO USAR PÁGINAS PARCIALES?

Una vista parcial permite agrupar un trozo de código de HTML, CSS, entre otros, que pueda ser reutilizado en otras vistas. En una vista podemos luego llamar mediante un método `Helpers` de la propiedad HTML para que agregue todo el código de una vista parcial. Una vista parcial es un trozo de interfaz de usuario que se puede reutilizar. Lo más común es disponerla en la carpeta `Shared`, y que su nombre comience con el carácter de subrayado.

VISTAS EN DJANGO

Una vista, generalmente conocida como una función en Python, hace una solicitud Web y devuelve una respuesta Web, que puede ser el contenido de una página, un error 404, una imagen, un documento XML, entre muchas cosas más. La vista contiene toda la lógica necesaria para devolver una respuesta, todas estas respuestas se encuentran en un único archivo llamado: `views.py`, que se encuentra dentro de cada aplicación de Django.

LAS PLANTILLAS Y LAS VISTAS

Las plantillas son muy importantes, permiten acomodar el resultado que devuelve la vista. Django tiene un estupendo motor de plantillas que permite separar eficientemente la presentación, de la lógica de programación.

HERENCIA DE PLANTILLAS

La herencia de plantillas es, por lo general, un conjunto de técnicas que se asimila a la programación orientada a objetos, en el que los bloques de contenido se insertan dentro de otras plantillas HTML. Permite crear una plantilla base que contiene todos los elementos comunes de un sitio, y define bloques que las plantillas secundarias pueden anular. En este sentido, podemos administrar partes de plantillas que heredan el contenido de una plantilla a otra, y también cambiar bloques de contenido. El uso de plantillas se vuelve simple y eficiente, ya que cada plantilla solo contiene las diferencias de la plantilla que extiende.

Por ejemplo, la plantilla base.html define un esqueleto HTML, que incluye todo el código HTML repetitivo común a todas las páginas, además de un encabezado y un pie de página. También define tres "ranuras" o "bloques" para que las plantillas "secundarias" las llenen:

- El título de la página.
- Un menú de navegación secundario.
- Un bloque de contenido.

Por ejemplo:

BASE.HTML

```
<html>
  <head>
    <link rel="stylesheeBIBLIOGRAFIAt" href="static/style.css">
    <title>{% block title %}{% endblock %}</title>
  </head>
  <body>
<nav class="primary-nav">
  <ul>
    <li><a href="#">Inicio</a></li>
    <li><a href="#">Contenido</a></li>
    <li><a href="#">Acerca de</a></li>
  </ul>
</nav>
```

```
    {% block secondNav %}{% endblock %}
    {% block content %}{% endblock %}
  </body>
</html>
```

Ahora, una página de plantilla secundaria `inicio.html`:

```
{% extends "templates/base.html" %}
{% block title %}Inicio{% endblock %}
{% block secondNav %}
<nav class="secondary-nav">
  <ul>
    <li><a href="#">Pagina 1</a></li>
    <li><a href="#">Pagina 2</a></li>
  </ul>
</nav>
{% endblock %}
{% block content %}
Hola, este es el contenido del block
{% endblock %}
```

Cualquier número de plantillas secundarias se puede encadenar en múltiples niveles de herencia. Puede mantener una plantilla base con toda la plantilla HTML, luego una segunda plantilla que podría definir un diseño de una sola columna, y una plantilla final que aloje el contenido.

Alternativamente, las plantillas pueden tener diferentes diseños heredados de la misma base, como una plantilla de dos columnas.