PROGRAMACIÓN WEB 2

Profesor(a):
Carlo Jose Luis Corrales Delgado

Estudiante:
Jorge Luis Mamani Huarsaya

Email:
jmamanihuars@unsa.edu.pe

Repositorio GitHub:
https://github.com/jorghee/django-projects

5 de junio, 2024

Idea fundamental sobre cómo se comporta Django

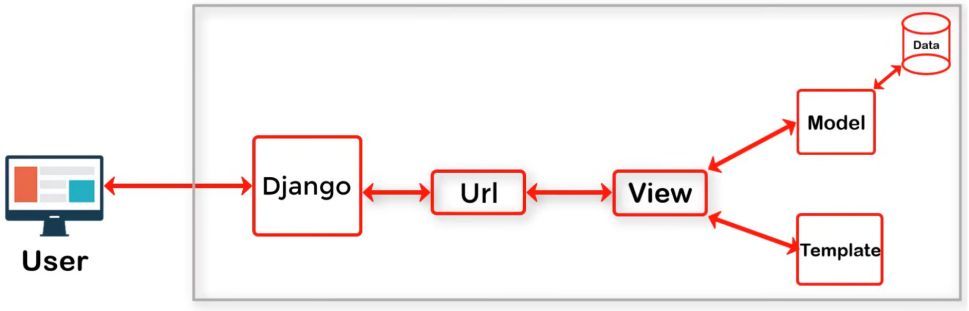

Figure 1: Model View Template

Agregando una plantilla HTML

Lo primero que deseamos hacer es informarle a Django que deseamos servir archivos como lo son archivos HTML que en Django está bien definida y separada de la lógica del proyecto. Entonces comenzamos modificando el archivo `settings.py` del proyecto para definir el directorio en el cual se almacenarán los archivos HTML que representan la presentación del proyecto. Nos vamos directamente a la variable `TEMPLATES`.

En la variable `TEMPLATES` agregamos un elemento al diccionario de clave `'DIRS'`

```
'DIRS': [os.path.join(BASE_DIR, 'templates')],
```

La constante `BASE_DIR` es la dirección hasta el directorio raiz del proyecto.

Una vez que tenemos configurada la dirección donde va a buscar Django las plantillas , podemos crear el directorio especificado y luego agregar un archivo `.html`. En este caso estamos utilzando una estructura ya definida y el archivo de inicio se denomina `index.html`. **Este archivo será el inicio de nuestra aplicación.**

Agregando una vista para manejar el archivo index.html

Siguiendo como es el funcionamiento de Django, ya hemos definido una Plantilla, por el momento no estamos definiendo trabajar con una base de datos, asi que nos toca definir **views**. Para ello nos posicionamos dentro de la Aplicación y modificamos el archivo `views.py`.

```
from django.shortcuts import render
from .models import Destination

def index(request):
  dests = Destination.objects.all()
  return render(request, "index.html", {'dests': dests})
```

Como se observa, estamos definiendo una función que se encarga de obtener una lista de objetos que posteriormente se explicará a detalle, pero lo importante aqui es que esta retornando el archivo `index.html`. Además se está enviando la lista de objetos que, porsupuesto, serán usados en el archivo `index.html`.

Mapeando las URLs en la Aplicación y en el Proyecto

Ya abordamos el manejo de una solicitud especifica, sin embargo Django no sabe cómo recibir las solicitudes y direccionar existosamente. Por lo tanto, necesitamos crear el archivo `urls.py` dentro de la Aplicación y hacer las redirecciones correspondientes.

```
from django.urls import path
from . import views

urlpatterns = [
  path("", views.index, name="index")
]
```

Como vemos, lo que se esta haciendo es establecer una ruta URL vacía (es decir, la página principal del proyecto) que será manejada por la vista index ubicada en el módulo views. El parámetro name="index" asigna un nombre a esta ruta, permitiendo que sea referenciada fácilmente en otras partes de la aplicación, como en plantillas HTML o redirecciones.

El archivo `urls.py` del proyecto

Necesitamos que Django sepa cómo enrutar las solicitudes entrantes a las vistas correspondientes

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
  path('', include('destinos_turisticos.urls')),
  path('admin/', admin.site.urls)
]
```

Con esta configuración hacemos que todas las URLs definidas en la Aplicación, es decir en el archivo `urls.py` de dicha aplicación, sean accesibles desde la raiz de nuestro sitio web.

Con estas modificaciones podemos lanzar nuestro servidor y nos daremos cuenta de que efectivamente el archivo `index.html` es encontrado, pero las hojas de estilos importadas y los scripts utilizados no son encontrados.

Entonces, ahora nos enfocamos en configurar un directorio especial que sirva **Archivos estáticos** como lo son los archivos .css, .js entre otros.

Configuración de archivos estáticos

Lo común en Django es crear un directorio nombrado como `static` donde se almacenarán todos los archivos estáticos, por lo tanto como hicimos anteriormente con el directorio `templates` necesitamos informarle a Django de esta configuración.

```
STATICFILES_DIRS = [
  os.path.join(BASE_DIR, 'static')
]
STATIC_ROOT = os.path.join(BASE_DIR, 'assets')
```

Estas modificaciones la hacemos en el archivo `settings.py` del proyecto. Por el momento como solo tenemos una aplicación, estamos proporcionando este directorio `static` global para el proyecto, sin

embargo una buena práctica es que cada Aplicación deba tener su propio directorio que almacene los archivos estáticos.

El directorio especificado por `STATIC_ROOT` es donde Django recopila todos los archivos estáticos de la aplicación al ejecutar collectstatic. Esto centraliza los archivos para que puedan ser servidos eficientemente en producción por un servidor web. En este caso, se define como assets en el directorio base del proyecto.

Modificando la dirección de los archivos importados en index.html

Para brindar correctamente la dirección de los recursos que necesita el archivo `index.html` es necesario realizar las siguiente modificaciones:

```
<link rel="stylesheet" type="text/css" href="{% static
↪  'styles/bootstrap4/bootstrap.min.css' %}">
```

El uso de `static` en el enlace es necesario para construir correctamente la URL de los archivos estáticos. Django reemplaza `{\% static \%}` con la URL base definida en `STATIC_URL`, asegurando que el navegador pueda encontrar y cargar el archivo CSS, independientemente del entorno (desarrollo o producción).

Definiendo Modelos usando el sistema de gestión de base de datos PostgreSQL

Hasta el momento no hemos utilizando en ningún momento Base de datos, sin embargo Django nos brinda una forma muy limpia de poder realizar operaciones en la base de datos sin la necesidad de usar el lenguaje `sql`.

Recordemos el modelo de trabajo de Django, ahora vamos a definir el concepto de `Model`. Para ello, nos vamos al archivo `models.py` de la Aplicación, y aqui podemos definir una clase como un modelo en Django. Un modelo en este framework es una clase que representa una tabla en la base de datos.

```
from django.db import models

class Destination(models.Model):
  name = models.CharField(max_length=100)
  img = models.ImageField(upload_to='pics')
  desc = models.TextField()
  price = models.IntegerField()
  offer = models.BooleanField(default=False)
```

- **name:** Un campo de texto (CharField) con un máximo de 100 caracteres.
- **img:** Un campo de imagen (ImageField) que almacena la ruta de la imagen en el directorio `pics`.
- **desc:** Un campo de texto largo (TextField) para descripciones.
- **price:** Un campo entero (IntegerField) para precios.
- **offer:** Un campo booleano (BooleanField) que indica si hay una oferta, con un valor predeterminado de `False`.

Lo interesante de este modelo es que todas las instancias de esta clase van a ser los destinos destinos turísticos de nuestro sitio web. Recordemos al momento de definir la función `index()` en el archivo `views.py` donde estuvimos obteniendo todos los objetos instanciados en la clase `Destination`, que básicamente estamos recuperando las filas de la base de datos.

Usando los objetos enviados al navegador en el archivo index.html

En la plantilla nosostros podemos iterar por la lista de objetos que se han enviando, este es una característica especifica de los sistemas de plantillas de Django, una funcionalidad valiosa.

```
{% for dest in dests %}
<div class="destination item">
  <div class="destination_image">
    <img src="{{dest.img.url}}" alt="">
    {% if dest.offer %}
    <div class="spec_offer text-center"><a href="#">Oferta especial</a></div>
    {% endif %}
  </div>
  <div class="destination_content">
    <div class="destination_title"><a
    ↪  href="destinations.html">{{dest.name}}</a></div>
    <div class="destination_subtitle"><p>{{dest.desc}}</p></div>
    <div class="destination_price">Desde S/. {{dest.price}}</div>
  </div>
</div>
{% endfor %}
```

Como vemos, estamos iterando sobre la lista de objetos, además estamos usando un condicional para agregar una etiqueta que informa si el tour esta en oferta o no. Recordemos que el campo `offer` está almacenda en la base de datos como un dato booleano.

Creando un Administrador

El Administrador debe tener la capacidad de poder modificar, insertar y eliminar destinos turísticos. Django proporciona una interfaz de administración incorporada que nos permite a nosotros los desarrolladores crear rápidamente una interfaz de administración básica para nuestras aplicaciones web sin necesidad de escribir código adicional.

Lo único que vamos a hacer es registrar el Modelo `Destination` en la interfaz de administración de Django.

```
from django.contrib import admin
from .models import Destination

admin.site.register(Destination)
```

Este registro habilita la administración básica de los objetos `Destination`, como la creación, edición y eliminación de registros, a través de la interfaz de administración de Django, que está disponible en la ruta predeterminada **/admin** del sitio web.