DESARROLLO WEB FULLSTACK

con Python y JavaScript

polotic misiones

polotic misiones

HTTP

¿Qué es HTTP?

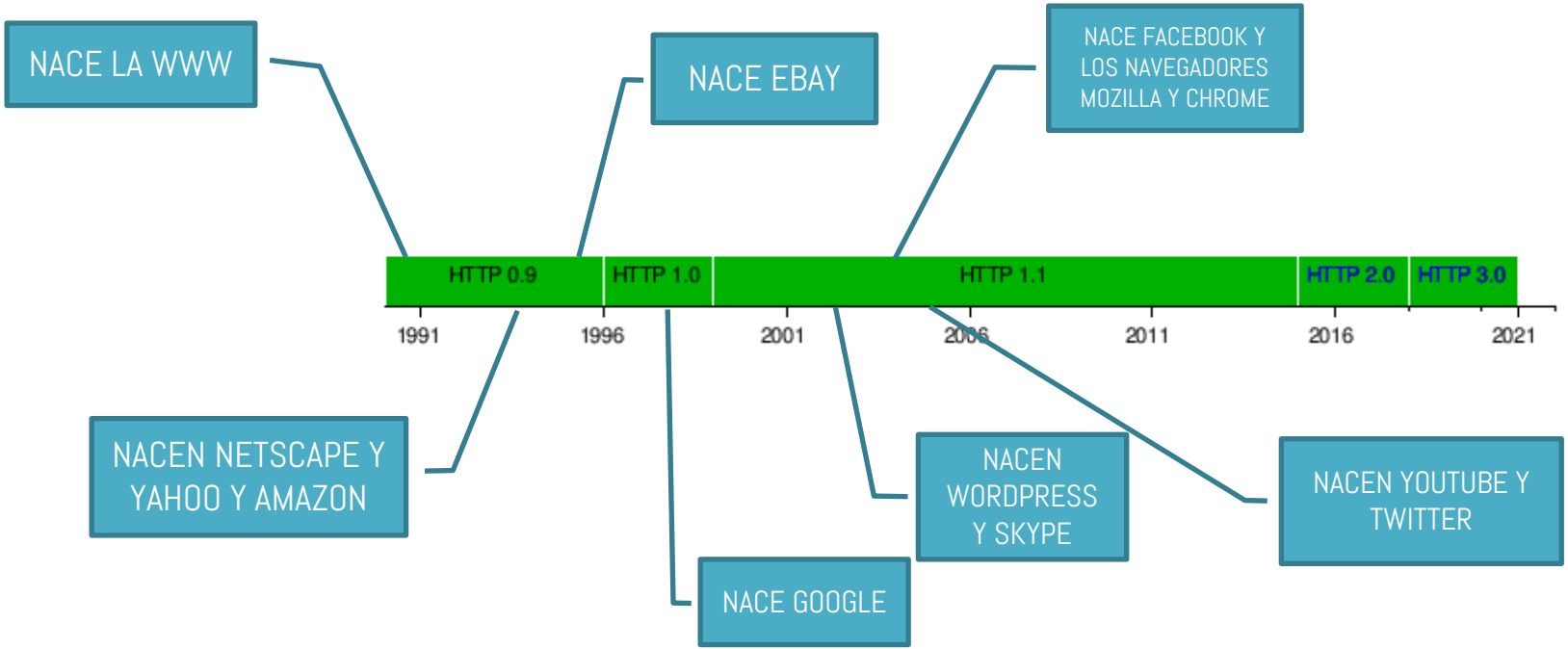
HTTP, o Protocolo de transferencia de hipertexto, es un protocolo ampliamente aceptado sobre cómo se transfieren los mensajes de un lado a otro a través de Internet.

Protocolo para la transferencia de varios formatos de datos entre servidor y cliente.

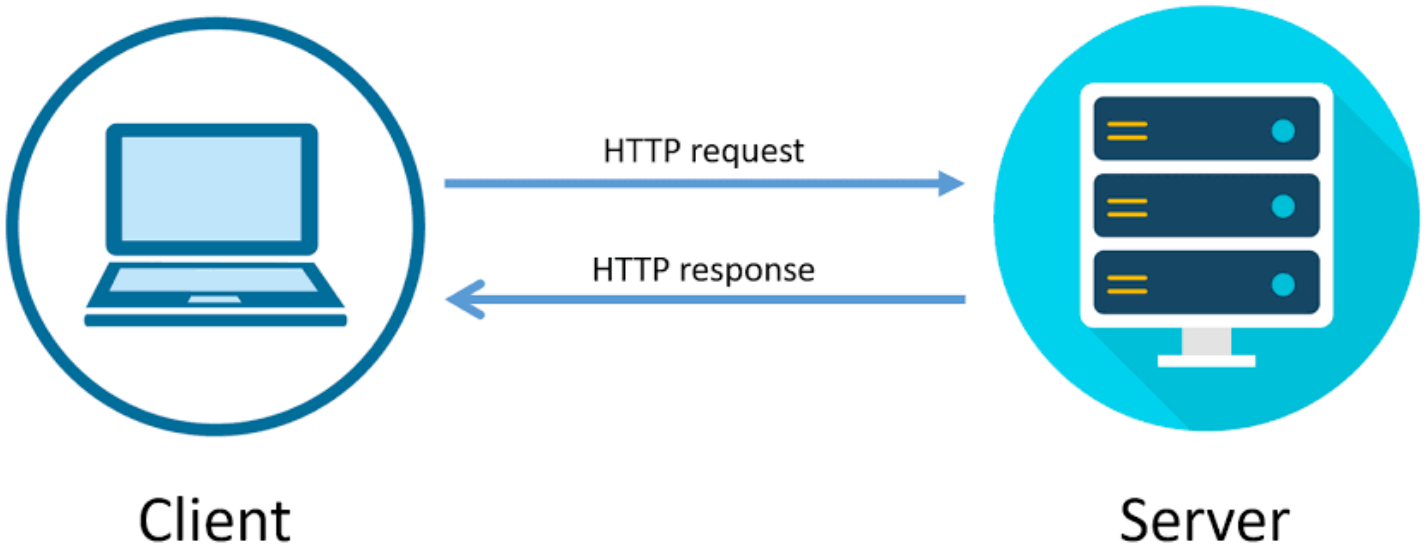
- Texto sin formato
- Hipertexto
- Imágenes
- Vídeo
- Sonido

También se transfiere meta-información

Versiones de HTTP



Comunicación por HTTP



Comunicación por HTTP - GET

```
GET / HTTP/1.1
Host: www.example.com
...
```

En el siguiente ejemplo, GET es simplemente un tipo de solicitud, uno de los tres mas comunes. Por lo general, la barra / indica que estamos buscando la página de inicio del sitio web y los tres puntos indican que también podríamos estar pasando más información.

Comunicación por HTTP – Respuesta

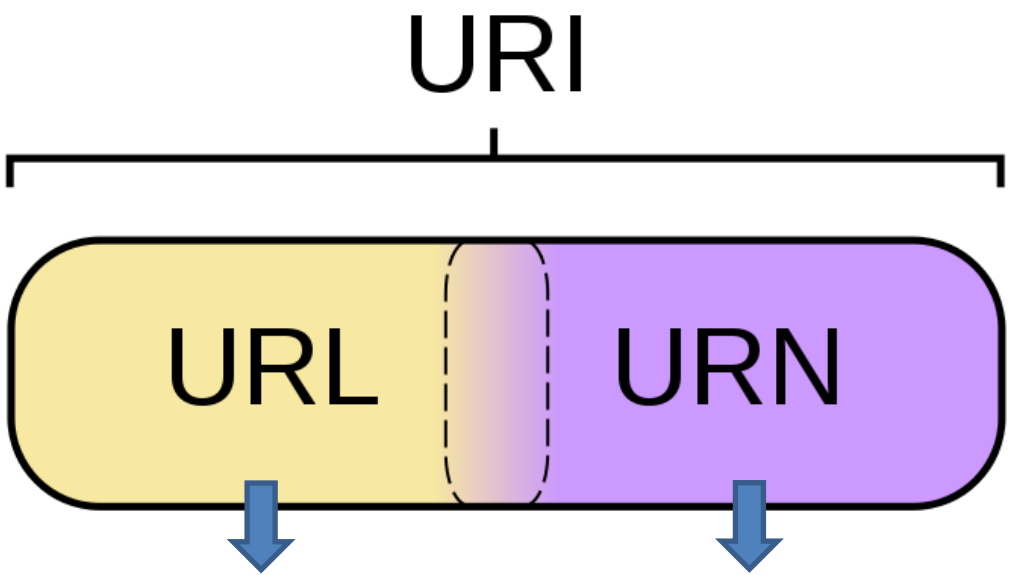
```
HTTP/1.1 200 OK
Content-Type: text/html
...
```

Después de recibir una solicitud, un servidor enviará una respuesta HTTP, que podría parecerse a lo siguiente. Dicha respuesta incluirá la versión HTTP, un código de estado (200 significa OK), una descripción del contenido y luego alguna información adicional.

Códigos de estado HTTP

Status Code	Description
200	OK
301	Moved Permanently
403	Forbidden
404	Not Found
500	Internal Server Error

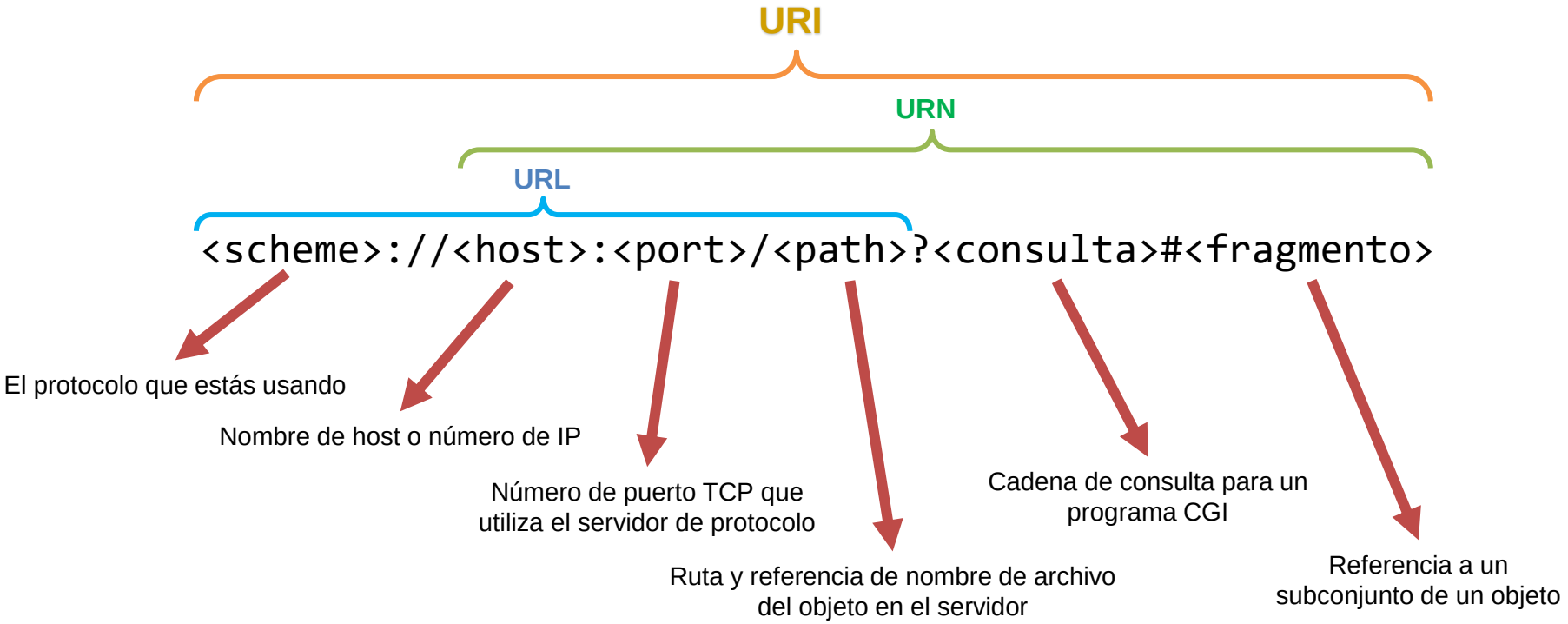
Estructura de los Recursos Web



Apunta a un documento en un servidor específico

Nombre de recurso uniforme

Estructura de los Recursos Web



URL y HTTP

- ✓ Todas las partes de la URL, excepto los parámetros, se utilizan con HTTP
- ✓ El esquema y el host se pueden omitir cuando el objeto referenciado está en la misma máquina que el documento de referencia
- ✓ El puerto se puede omitir siempre que el host al que se hace referencia se esté ejecutando en el puerto listado en tu archivo `/etc/services` del sistema operativo
- ✓ Normalmente puerto 80
- ✓ **Ruta completa** utilizada cuando se hace referencia a otro servidor
- ✓ **Ruta relativa** en el mismo servidor

La referencia con la ruta relativa es una URL parcial

polotic misiones

django

¿Qué es Django?

- Django es un framework Python de alto nivel que fomenta un desarrollo rápido de aplicaciones web, en conjunto con un diseño limpio y pragmático.
- Nos permitirá escribir código Python que genere HTML y CSS de forma dinámica.
- La ventaja de usar un framework como Django es que ya hay mucho código escrito para nosotros que podemos aprovechar.

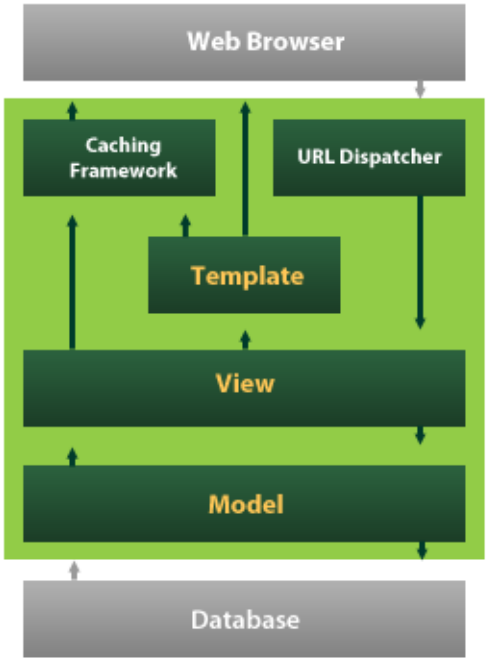

django
HTML
CSS
JS

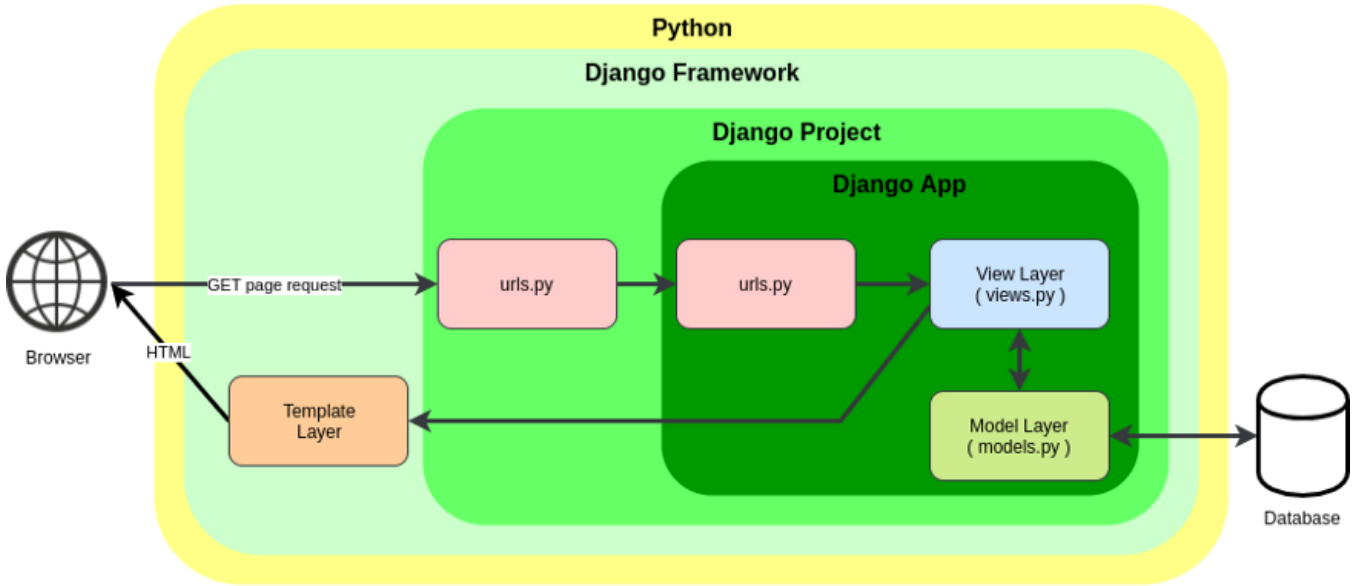
Arquitectura de Django

Arquitectura MVT:

- ***Modelos***
  - ❖ Describe tu estructura de datos (esquema de base de datos)
- ***Views***
  - ❖ Controla **que** un usuario visualiza
- ***Templates***
  - ❖ Controla **como** un usuario visualiza
- ***Controlador***
  - ❖ El Framework Django
  - ❖ Parseo de URLs

Arquitectura de Django





¿Qué ofrece Django?

- ✓ Interfaz de administración (CRUD)
- ✓ Sistema de autenticación
- ✓ Manipulación de formularios
- ✓ Sesiones
- ✓ Internacionalizacion
- ✓ Cacheo

¿Qué ofrece Django?

- ✓ Mapeador relacional de objetos
- ✓ Diseño de URL elegante
- ✓ Sistema de plantillas/templates

Principios

- ✓ Patron de diseño MVC
- ✓ DRY – Don´t repeat yourself
- ✓ Principio de bajo acoplamiento
- ✓ Separacion de la lógica de negocios del HTML

Instalando Django

Para comenzar, tendremos que instalar Django, lo que significa que también tendrás que tener instalado `pip` (gestor de paquetes de Python)

Instalas Django con el siguiente comando:

```
pip3 install Django
```

Creando un proyecto Django

Después de instalar Django, podemos seguir los pasos para crear un nuevo proyecto Django:

```
django-admin startproject NOMBRE_PROYECTO
```

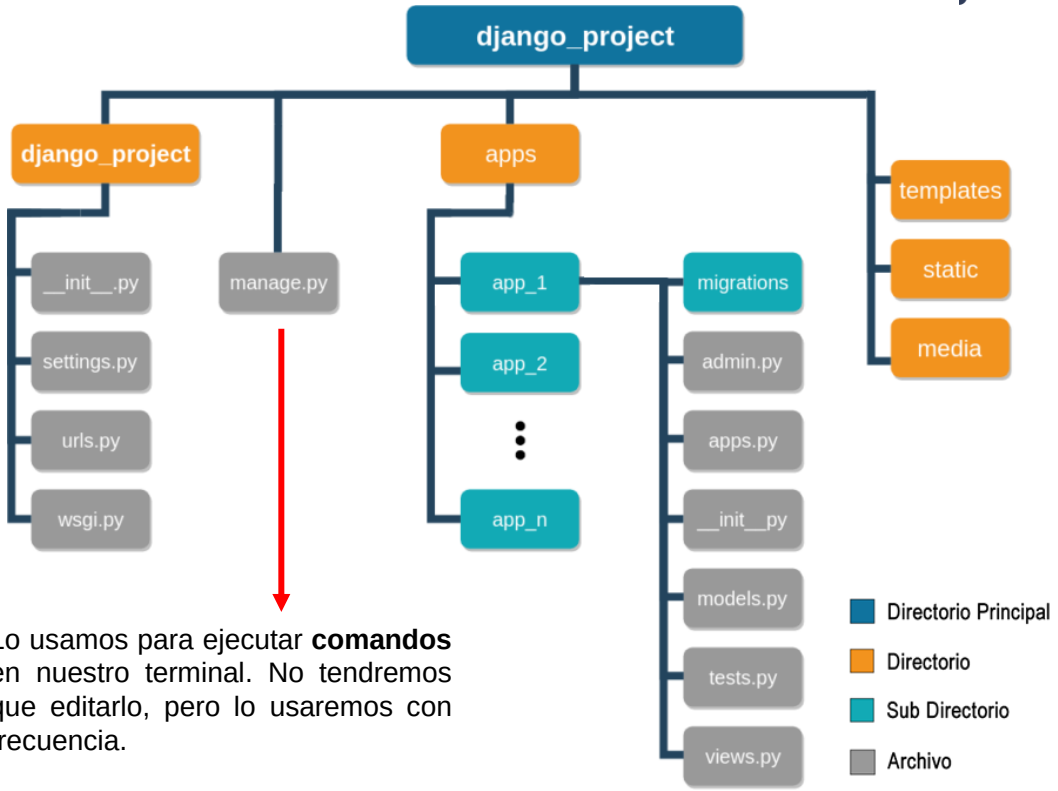
Luego tendrás un directorio `NOMBRE_PROYECTO` con todos los archivos generados.

Estructura

Contiene algunos **ajustes** de configuración importantes.
Es posible que deseemos cambiar algunos ajustes predeterminados de vez en cuando.

Contiene instrucciones sobre a dónde se debe dirigir a los usuarios después de navegar a una determinada **URL**.

Lo usamos para ejecutar **comandos** en nuestro terminal. No tendremos que editarlo, pero lo usaremos con frecuencia.



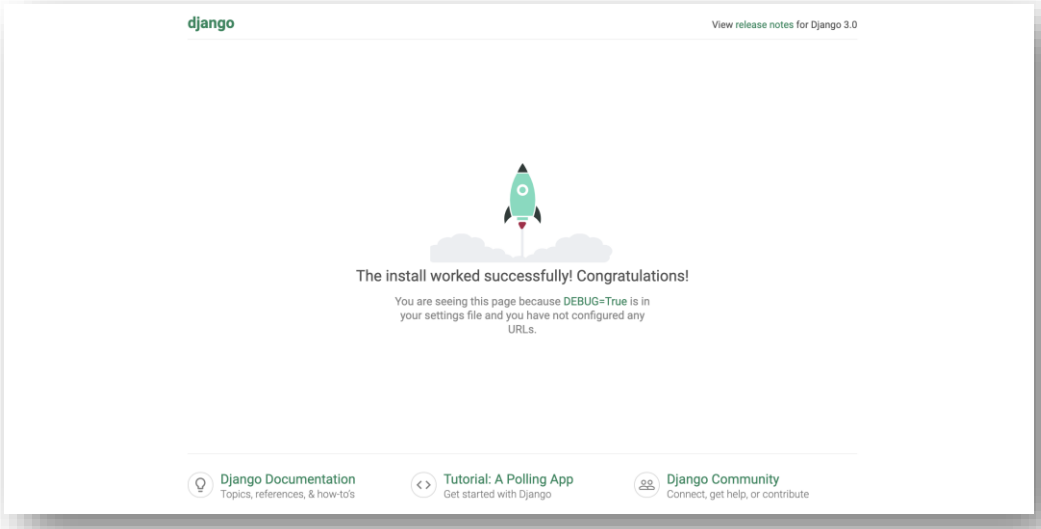
Inicializar el Servidor

Después de instalar Django y crear nuestro primer proyecto, podemos levantar el servidor mediante el siguiente comando:

```
python manage.py runserver
```

Inicializar el Servidor

Este servidor de desarrollo **se ejecuta localmente** en tu computadora, lo que significa que otras personas no podrán acceder a tu sitio web. En el output te mostrará una URL que debería llevarte a una página de destino predeterminada:



Crear una Aplicación Django

Los proyectos en Django se dividen en aplicaciones (apps) fomentando así la modularidad en el desarrollo de software. Luego de crear el proyecto, creamos la aplicación, que es donde desarrollaremos nuestra lógica de negocios mediante el siguiente comando:

```
python manage.py startapp NOMBRE_APP
```

Instalar la Aplicación Django

Pero esto aún no termina aquí.

Tenemos que **vincular** la aplicación con el proyecto previo. Para ello nos dirigimos al archivo `settings.py` de nuestro proyecto y en la sección `INSTALLED_APPS` agregamos la aplicación:

```
# Application definition

INSTALLED_APPS = [
    'myapp',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Views

Ahora sí, tenemos todo para comenzar a desarrollar nuestra aplicación web con Python.

Lo primero que hacemos es dirigirnos al archivo `views.py` donde crearemos una vista para que pueda verse nuestra aplicación. Podemos pensar la vista como una página donde el usuario podrá acceder.

Views

Por ahora solamente crearemos una respuesta HTTP 200 con una cadena de texto que represente un mensaje personalizado:

Importamos la funcionalidad que dibuja una pagina web

Importamos la funcionalidad que nos permite realizar comunicaciones usando el protocolo HTTP

Los nombres de las funciones estarán relacionados al tipo de página que renderizan. En éste caso el índice.

Retorno un string al HttpResponse que será convertido automáticamente a HTML.

```python
from django.shortcuts import render
from django.http import HttpResponse

# Crea tus vistas aqui
def index(request):
    return HttpResponse("¡Hola, mundo!")
```

Rutas

Pero la vista anterior está aislada, y necesitamos poder referenciarla desde Django. Es por ello que existen las rutas URL. Éstas rutas se definen en el archivo `urls.py` que se encuentre en el mismo directorio que `views.py`

Ya tenemos un archivo `urls.py` para todo el proyecto, pero es mejor tener uno separado para cada aplicación individual.

Rutas

Pero la vista anterior está aislada, y necesitamos poder referenciarla desde Django. Es por ello que existen las rutas URL. Éstas rutas se definen en el archivo `urls.py` que se encuentre en el mismo directorio que `views.py`

Ya tenemos un archivo `urls.py` para todo el proyecto, pero es mejor tener uno separado para cada aplicación individual.

Rutas – URLs para la Aplicación

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

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

Importar nuestro archivo views.py

Ruta de la URL

Función a la que acceder en views.py

Opcionalmente le podemos dar un nombre único para esa ruta

Rutas – URLs para todo el Proyecto

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

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

Admin nos permite acceder al gestor administrativo de nuestro proyecto

El include permite redireccionar hacia la aplicación desde el proyecto

Rutas

Ahora podemos acceder a nuestra aplicación escribiendo la URL

`http://localhost:8000/nuestra_app`

El sistema buscará el path “`nuestra_app`” dentro del `urls.py` del proyecto que lo enviará al `urls.py` de la aplicación. Allí dentro se buscará automáticamente el directorio raíz (porque no proporcionamos ningún path) que llevará al la función `index` del `views.py` de la aplicación.

Múltiples Vistas

Ahora, si queremos, podemos cambiar la función `index` dentro de `views.py` para que devuelva lo que queramos. Incluso podríamos realizar un seguimiento de las variables y hacer cálculos dentro de la función antes de devolver algo.

Por ejemplo, vamos a hacer algo simple, dentro de nuestra aplicación vamos a crear varias páginas que saluden, por ejemplo a Harry y a Hermione.

Dentro de `views.py`:

```python
from django.shortcuts import render
from django.http import HttpResponse

# Crea tus vistas aqui.
def index(request):
    return HttpResponse("¡Hola, mundo!")

def harry(request):
    return HttpResponse("¡Hola, mundo!")

def hermione(request):
    return HttpResponse("¡Hola, mundo!")
```

Múltiples Vistas

Dentro del `urls.py` dentro de nuestra aplicación:

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

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

URL parametrizadas

Muchos sitios web están parametrizados por elementos incluidos en la URL. Por ejemplo, ir a *https://www.facebook.com/poloticmisiones/* te mostrará todas las publicaciones de la página del PoloTIC, y al ir a *https://www.instagram.com/poloticmisiones/* verás el muro de fotos e imágenes del PoloTIC. Incluso puedes encontrar tu propio muro de Instagram navegando a *https://www.instagram.com/TU_USUARIO*

Al pensar en cómo se implementa esto, parece imposible que sitios como Facebook e Instagram tengan una ruta URL individual para cada uno de sus usuarios, así que veamos cómo podríamos hacer una ruta un poco más flexible. Comenzaremos agregando una función más general, llamada `saludar`, a `views.py`

```
def saludar(request, nombre):
    return HttpResponse(f"Hola, {nombre}!")
```

URL parametrizadas

Esta función acepta no solo una solicitud, sino también un argumento adicional del nombre de un usuario y luego devuelve una respuesta HTTP personalizada basada en ese nombre.

A continuación, tenemos que crear una ruta más flexible en `urls.py`, que podría verse algo así:

```
path("<str:nombre>", views.saludar, nombre="saludar")
```

Esta es una sintaxis nueva, pero esencialmente lo que está sucediendo aquí es que ya no buscamos una palabra o nombre específico en la URL, sino cualquier cadena que un usuario pueda ingresar.

URL parametrizadas

Incluso puedo hacer que estos saludos se vean un poco más agradables, aumentando la función de saludo para utilizar la función de capitalización de Python que capitaliza una cadena:

```
def saludar(request, nombre):
    return HttpResponse(f"Hola, {nombre.capitalize()}!")
```

Esta es una gran ilustración de cómo cualquier funcionalidad que tenemos en Python se puede usar en Django antes de ser devuelta a la vista HTML.

Emitir HTML desde Django

- Django es un framework Python que permite utilizar plantillas para formatear la manera en que se muestra el contenido en el navegador web.
- Hasta ahora solo habíamos hecho que las funciones de las vistas retornen solamente un mensaje HTTP. Sin embargo podemos incluir el contenido HTML que querramos.

Emitir HTML desde Django

- Puedo decidir devolver un encabezado azul en lugar de solo el texto en mi función index:

```
def index(request):
    return HttpResponse("<h1 style=\"color:blue\">¡Hola, mundo!</h1>")
```

Utilizando Plantillas HTML en Django

Sería muy tedioso escribir una página HTML completa dentro de `views.py`. También constituiría un mal diseño, ya que queremos mantener partes separadas de nuestro proyecto en archivos separados siempre que sea posible.

Es por eso que ahora incorporamos las plantillas (templates) de Django a nuestro proyecto.

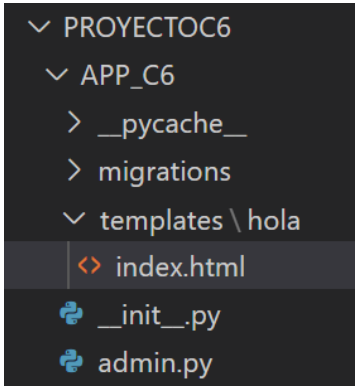
Estas plantillas nos permitirán escribir código HTML y CSS en archivos separados y renderizar esos archivos usando Django.

Utilizando Plantillas HTML en Django

La sintaxis que usaremos para renderizar una plantilla es la siguiente:

```
def index(request):
    return render(request, "hola/index.html")
```

Sin embargo no tenemos la plantilla index.html. Así que tenemos que crearla. Para ello creamos una carpeta `templates` dentro de nuestro directorio de la aplicación. Y dentro de `templates` creamos un directorio `hola` y allí dentro el archivo `index.html`



Escribiendo HTML en Django

Ahora podemos escribir código HTML en nuestro archivo index.html

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Hola</title>
    </head>
    <body>
        <h1>¡Hola, mundo!</h1>
    </body>
</html>
```

Y si vamos a la página principal de nuestra aplicación podemos ver éste código renderizado.

Django Templates - Plantillas

Además de escribir algunas páginas HTML estáticas, también podemos usar el lenguaje de plantillas de Django para cambiar el contenido de nuestros archivos HTML según la URL visitada. Probémoslo cambiando nuestra función de saludo anterior:

```
def saludar(request, nombre):
    return render(request, "hola/saludar.html", {
        "nombre": nombre.capitalize()
    })
```

Observa que pasamos un tercer argumento a la función de renderización aquí, uno que se conoce como **contexto**. En este contexto, podemos proporcionar información que nos gustaría tener disponible dentro de nuestros archivos HTML. Este contexto toma la forma de un **diccionario Python**.

Django Templates - Plantillas

Ahora, podemos crear un archivo `saludar.html`:

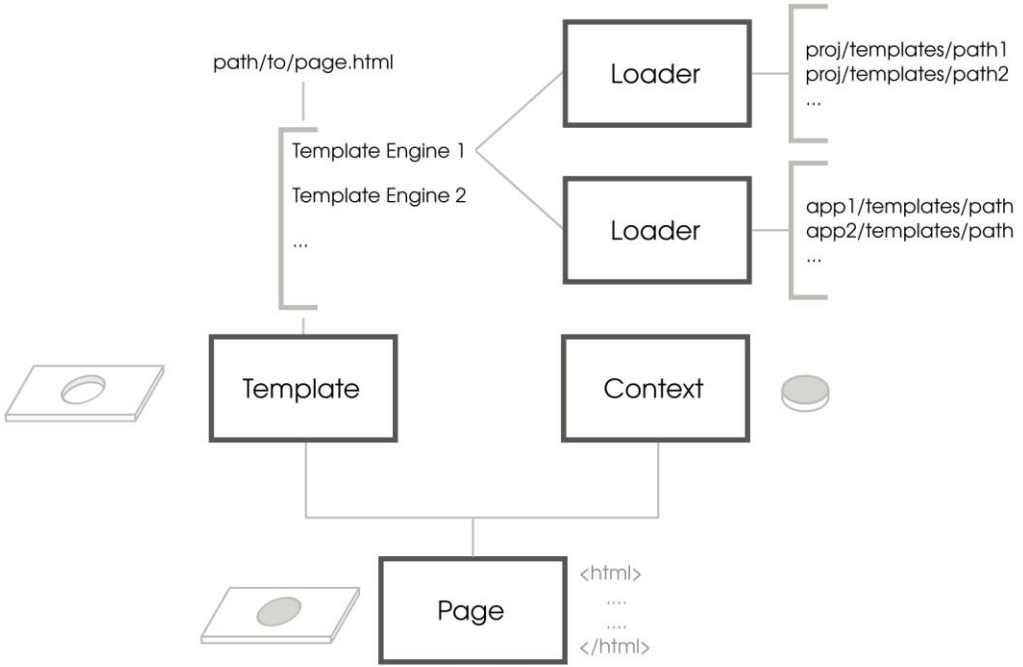
```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Hola</title>
    </head>
    <body>
        <h1>Hola, {{ nombre }}!</h1>
    </body>
</html>
```

Notarás que usamos una nueva sintaxis: **llaves dobles**. Esta sintaxis nos permite acceder a las variables que hemos proporcionado en el argumento de contexto.

Django Templates - Plantillas

Ahora, hemos visto cómo podemos modificar nuestras plantillas HTML según el contexto que proporcionamos.

Sin embargo, el lenguaje de plantillas Django es aún más potente, así que echemos un vistazo a algunas otras formas en que puede ser útil.



Django – Estilos CSS

Si queremos agregar un archivo CSS (que podríamos considerarlo como un archivo estático porque no cambia durante el tiempo de ejecución de la aplicación) primero debemos crear una carpeta llamada `static.`

Luego crearemos una carpeta de `anionuevo` dentro de esa, y luego un archivo `styles.css` dentro de ella.

En este archivo, podemos agregar cualquier estilo que deseemos tal como lo hicimos anteriormente:

```
h1 {
    font-family: sans-serif;
    font-size: 90px;
    text-align: center;
}
```

Django – Estilos CSS

Ahora, para incluir este estilo en nuestro archivo HTML, agregamos la línea `{load static}` en la parte superior de nuestra plantilla HTML, que le indica a Django que deseamos tener acceso a los archivos en nuestra carpeta static. Luego, en lugar de programar el enlace a una hoja de estilo como hicimos antes, usaremos una sintaxis específica de Django:

```
<link rel="stylesheet" href="{% static 'anionuevo/styles.css' %}">
```

Sin embargo, aunque recarguemos la pagina en nuestro navegador no veremos resultados. Para ello debemos reiniciar el servidor Django.

Django – Herencia en Templates

En el `layout_padre.html` debes definir donde iran los hijos con el tag:

```
{% block body %}
{% endblock %}
```

Observa que nuevamente hemos usado {% ...%} para denotar algún tipo de lógica no HTML, y en este caso, le estamos diciendo a Django que llene este "bloque" con texto de otro archivo.

En el template hijo debes declarar la cabecera:

```
{% extends "tuaplicacion/layout_padre.html" %}
```

Y en el cuerpo pondrás los limites superior e inferior delimitados en el padre, y dentro de ellos el código propio del hijo:

```
{% block body %}
...
{% endblock %}
```

Trabajemos Juntos

Llegó la hora del desafío:

- Intenta construir tu propia aplicación en Django, donde puedas visualizar un portal web con una cabecera, un menú, una sección donde colocar artículos y un pie de página.

¡HASTA LA *próxima!*

www.polotic.misiones.gob.ar

/poloticmisiones

polotic misiones

Gobierno de Misiones