Curso web de Django

Píldoras informáticas

Notas

Es una herramienta de software que utiliza frameworks lo que te permite crear sitios webs (complejos) de formas rápida y sencilla.

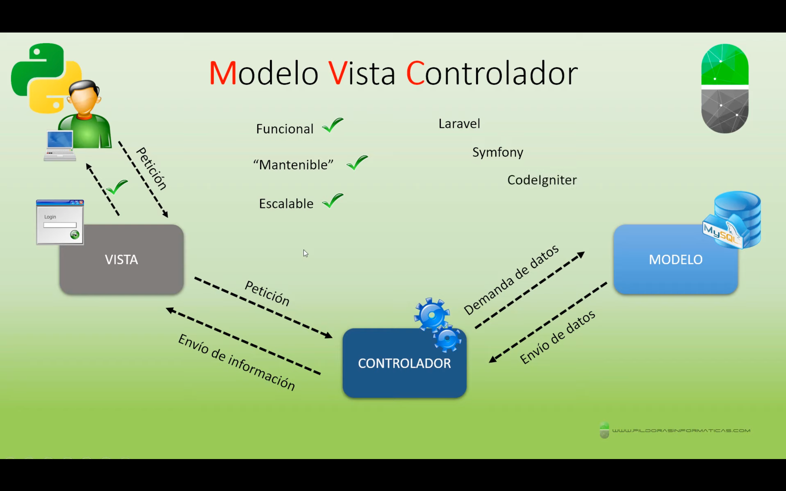
Se pueden utilizar los mimos pedazos de código para crear diferentes sitios webs

Se basa en un patrón MVC (Modelo Vista Controlador o Model View Controller)

Este patrón consiste en dividir cualquier aplicación en tres grandes módulos:

* **Modelo:** Se encarga de gestionar los datos obtenidos de una base de dato
* **Vista:** Mostrar la info al usuario, lo que este ve y con lo que interactúa
* **Controlador:** Encargado de gestionar las comunicaciones ente la vista y el modelo por medio de peticiones (ej. Un formulario web – lo cual seria la vista) esta petición la recibe el controlador y demanda datos al modelo.

Ej. El usuario hace la petición de reservar un vuelo o una habitación de hotel, el controlador demanda los datos necesarios al modelo (vuelos disponibles, horarios, habitaciones vacías, tiempo de estadías, etc.) El modelo le envía estos datos al controlador, quien le envía esto a la vista, lo que lo permite ser visualizado por el usuario.



Django lo llama MTV (Model template view), pero la filosofía o idea general es la misma.

Tipos de instalación:

**Local:**

* Única versión de Django, la que tengas instalada
* Única versión de Python
* Mismas dependencias para todos los proyectos que hagas

**Virtual:**

* Un desarrollo mas serio
* Varias versiones de Django
* Varias versiones de Python
* Diferentes dependencias
* Diferentes dependencias para todos los proyectos que hagas
* Te permite igualar los entornos de desarrollo – pruebas – producción.
  + Desarrollo: proyecto en construcción, solo pueden verlo los desarrolladores
  + Pruebas: si esta publicado, pero aun esta en fase de test, pruebas, cambios, y modificaciones
  + Producción: Ya es publico, y es visible par todo el mundo.

**Gestores de bases de datos (BBDD):**

* SQLite 3: gestor BBDD por defecto
* PostgreSQL: Gestor recomendado por Djangop
* MySQL
* Oracle

**Conectores ofrecidos por terceros:**

* SQL server
* SAP SQL
* DB2
* Firebird

Para crear el primer proyecto en Django:

**(Todo en el Terminal)**

**(ojo con los espacios, las mayúsculas y las minúsculas)**

* Crea una carpeta dedicada a ese proyecto
* cd (Copiar dirección de esa carpeta)
* django-admin startproject [nombre del Proyecto]

**Se debe de crear una base de datos con sqlite3:**

**(en la consola/terminal)**

* python manage.py migrate
* python manage.py runserver

**Copy the server in a search engine**

Ej:

Starting development server at http://127.0.0.1:8000/

Copy

127.0.0.1:8000

**Los MTV:**

**Model:** Base de datos, es demandado datos y este los envia

**Template:** Interface de ionteraccion con el usuario, recibe la informacion y hace peticiones al view

**View:**  Demanda los datos de acuerdo a las peticiones del template

**Django trabaja con dos objetos fundamentales:**

* **Clase Request:** Para hacer peticiones
* **HttpResponse:** para enviar la informacion o la respuesta

**Camino URL**

Cuando queremos crear una respuesta a un request, primero se debe de crear una funcion que retorne una respuesta http. (HttpResponse)

Ej:

﻿def despedida(request):

return HttpResponse("It was a pleasure to have you on this website! See you soon!")

Luego, para esta funcion, hay que crear una url, vamos al documento.py de urls (urls.py), y la creamos:

urlpatterns = [

path('admin/', admin.site.urls),

path('despedida/', despedida),

]

Lo amarillo es la url, lo que vamos a poner el el buscador ( ej. localhost:8000/despedida/), esta puedes ponerle como quieras.

Lo fucsia es la función a llamar, es decir que debe de ser idéntica a la función que creaste anteriormente.

Es importante no olviodar importar dicha funcion previamete de manera:

From Parent\_module.Document\_where\_function\_was\_created import function

Ej:

﻿from Proyecto1.views import despedida

**Parámetros en URL**

Paso de parámetros a través de la URL

Muy común con sitios web dinámicos, es necesario y frecuente

NOTA: Para abrir el servidor:

Python manage.py runserver

Podemos hacer que la respuesta del servidor este en formato **html:**

def despedida(request):

return HttpResponse("<html><body><h1>It was a pleasure to have you here!<h1/><body/><html/>”)

El tipo **httpResponse** puede tomar una variable, es decir que podemos:

Tipo\_html = "<html><body><h1>It was a pleasure to have you here!<h1/><body/><html/>”

o tambien:

Tipo\_html = “””<html>

<body>

<h1>

It was a pleasure to have you here!

<h1/>

<body/>

<html/>”””

Y luego:

def despedida(request):

return HttpResponse(Tipo\_html)

Django utliza un URL Friendly, es decir URL sencillas para que sea sencillo buscarlos, se intenta evitar al utliliozar parametros en la URL simbolos (como ¿?) que se utiliza mucho en html

**Para crear una URL la cual pueda recibir un parametro:**

path(‘calculaEdad/<parametro>, funcion\_a\_llamar)

ej:

path(‘calculaEdad/<int:agno>’, EdadFutura)

**en el buscador:**

http://localhost:8000/calculaEdad/2050

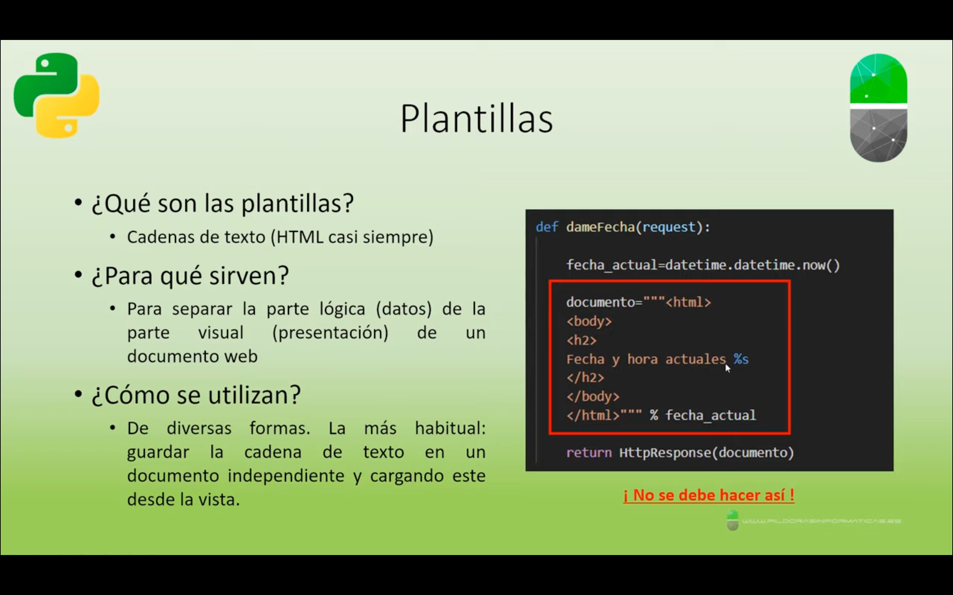
Todo en una URL es considerado como texto, por esta razón se utiliza el int: antes de el parámetro de edad <agno>

Se pueden añadir todos las entradas que quieras a la URL:

path(‘calculaEdad/<int:edad>/<etc>/<etc>/<int:agno>’, EdadFutura)

**Las Plantillas:**

* ¿Que son?
  + Son cadenas de texto que pueden contener código html o cualquier documento de texto plano
* ¿Para que sirve?
  + Para separar la parte lógica del proyecto en Django (los datos) y de la parte visual, el formato, es decir lo que el usuario ve.
  + Ej: el código html iba incrustado en las vistas, lo que es una mala practica. Únicamente se justifica en momentos de crear paginas web muy sencillas.
  + Nos van a permitir separar el código html (el formato del documento to web) de la lógica (el código en Python)



Consiste en guardar la cadena de texto, es decir desde la etqueta de apertura <html> hasta la etiqueta de cierre </html>.

Guardar esa cadena de texto en un documento aparte e independiente, en una carpeta que tu quieras, y luego cargar este documento en la vista.

* **Ventajas:**
  + Se pueden hacer cambios en el diseño con independencia del código (o la lógica) de la vista
  + Se puede diversificar el trabajo, al trabajar muchas personas, unas pueden encargarse de la interface (lo que el usuario ve) y los programadores se encargan de la lógica o del contenido.
  + Si el código html esta incrustado en la lógica, es muy difícil que varias personas trabajen en un mismo proyecto simultáneamente.

**¿Cómo?:**

* Crear un objeto tipo Template:
  + plt = Template(doc\_externo.read())
* Crear un contexto:
  + Son los datos adicionales que puede llegar a utilizar un template o documento html, por ejemplo el contenido dinamico (edad, fecha y hora). Y esto se almacena en el contexto.
  + Si no se almacena contenido dinamico, igual se debe crear un contexto, aunque este quede vacio.
  + ctx=context()
* Renderizar el objeto template:
  + Documento=plt.render(ctx)
  + Llamar a este método, y pasarle como parámetro el contexto (aunque este vacío)

**Variables en las plantillas:**

* Se debe de utilizar un diccionario dentro del contexto

**Llamada a métodos desde las plantillas:**

* **Jerarwquia u ordn en llamadas desder las plantillas**

**Uso de liostas en los contextops y en las plantillas:**

**Estructuras de control de flujo en las plantillas:**

**Jerarquía u orden de llamada dentro de las plantillas:**

1. Diccionario
2. Atributo
3. Método
4. Índice de lista

**Para abrir un bucle for en django:**

<p>

{% for i in list %}

<p>{{i}}</p>

{% endfor %}

</p>

**Otras opciones:**

<p>

<ul> *#lista desordenada*

{% for i in list %}

<li>{{i}}</li>

{% endfor %}

</P>

El resultado de esto sera:

* i
* i+1
* i+2
* i+n

dependiendo del len(list)

**para un condicional:**

<p>

<ul>

{% if list %}

{% for i in list %}

<li>{{i}}</li>

{% endfor %}

{% else %}

<p> No hay elementos </p>

{% endif %}

</P>

**OJO:**

**Recuerda que el uso de .**etc **es para llamar a propiedades y el uso de** .etc() **es para llamar a métodos, en Djano no se utilizan los paréntesis en las llamadas a métodos, ej;** .upper() **seria** .upper