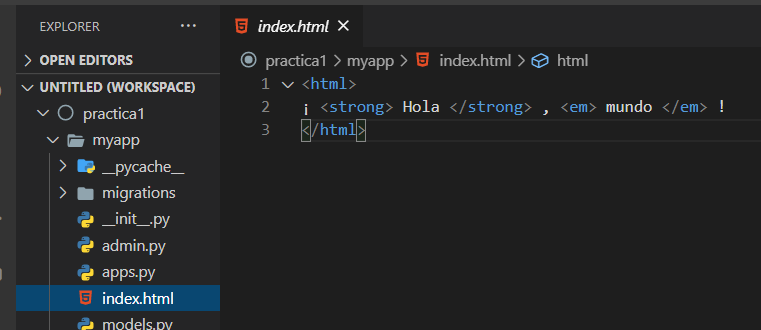
**MANUAL DEL ALUMNO**

Una buena practica, no mezclar lo lógico (Python) con lo grafico (html). Pag 1. Por eso en lugar de devolver un html en la respuesta podríamos llamar a un html aparte creandolo.

Creamos un nuevo proyecto llamado **practica1** (django-admin startproject practica1) con una aplicación **myapp** (python manage.py startapp myapp). Dentro de la carpeta de **myapp** colocamos el siguiente index.html



Codigo:

<html>

    <strong>Hola</strong> , <em>mundo</em>!

</html>

Partir del siguiente código:



b. Creamos la siguiente vista / función en myapp/views.py:

from django.shortcuts import render

from django.http import HttpResponse

def index ( request ):

    f = open ( "myapp/index.html" , encoding = "utf-8" )

    response = HttpResponse(f.read())

    f.close()

    return response

c. myapp/urls.py:

from django.urls import path

from . import views

urlpatterns = [

path( "" , views.index, name = "index" ),

]

d. urls.py (del proyecto):

from django.urls import path, include

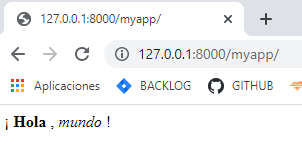
urlpatterns = [

    path("myapp/", include("myapp.urls"))

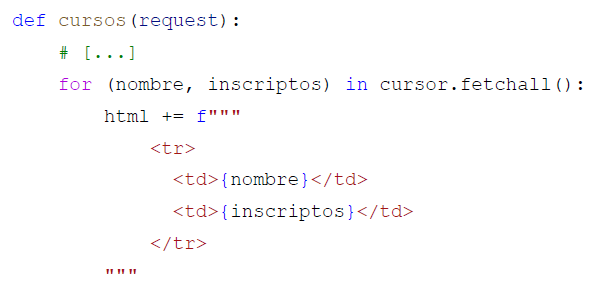
]

e. Migramos (python manage.py migrate) e iniciamos el servidor (python manage.py runserver).

f. <http://127.0.0.1:8000/myapp/>



Con este procedimiento si bien logramos separar la lógica del entorno grafico la desventaja es que perdemos la posibilidad de generar código HTML a partir de datos de Python, como figura en el siguiente ejemplo, vista cursos:

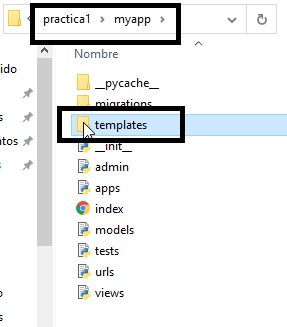


Para solucionar este problema Django incluye el sistema de **Plantillas**.

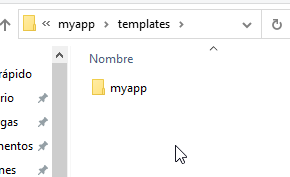
**Plantillas (Templates)**

Permite crear código HTML incorporando pequeñas porciones de código Python. En el caso de Django permite incorporar un dialecto diseñado para este propósito, **Django Template Languaje.**

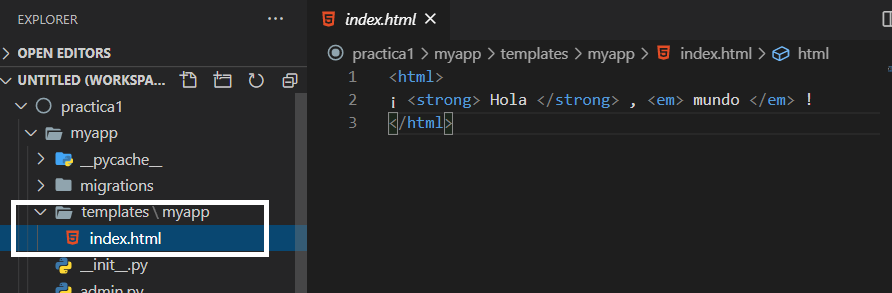
1.Creamos una carpeta llamada **templates** dentro de nuestra aplicación **myapp:**



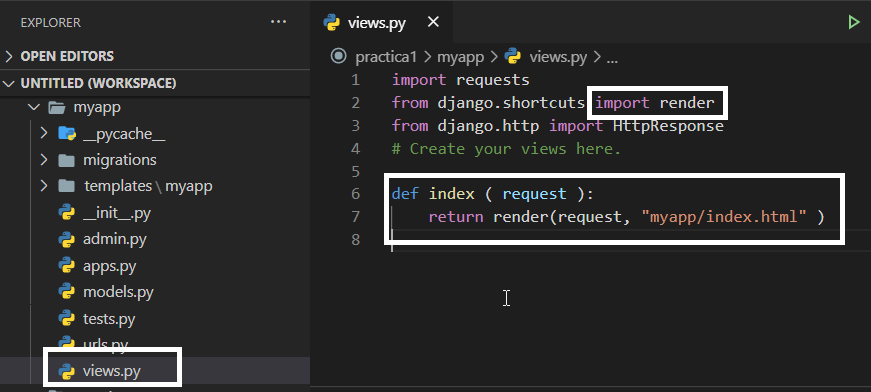
2. Dentro de la misma creamos la carpeta **myapp:**

****

3. Dentro de **myapp** colocamos el archivo **index.html** creado previamente (por ahora todo manual):



4. Cambiamos nuestra función **index()** en **views.py** por la siguiente:

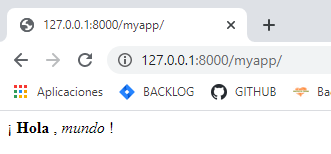


render(): recibe como 1er parámetro el request de la vista que va a ser renderizada. El request y el nombre del template son los parámetros minimos. Las **vistas** son funciones que obtienen un **request** y restornan una respuesta o **response**. Dentro del **url.py** (de la aplicación)llamamos a las funciones correspondientes gracias al **urlpattern**.

5. Recordamos setear en **settings.py** en **INSTALLED\_APPS** la correspondiente aplicación:

    'myapp.apps.MyappConfig'

5. <http://127.0.0.1:8000/myapp/>



Obtenemos el mismo resultado que el procedimiento anterior. Pero en este caso lo importante es que **el código HTML se esta cargando desde el archivo myapp/templates/myapp/index.html**.

render() lee el contenido del archivo, lo procesa con el sistema de plantillas de Django (sublenguaje), y devuelve el resultado como una instancia de django.http.HttpResponse.

**Codigo Resultante:**

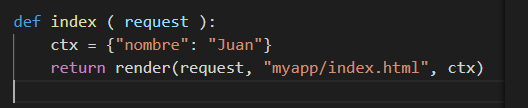


**FUNCIONALIDADES DEL SISTEMA DE PLANTILLAS**

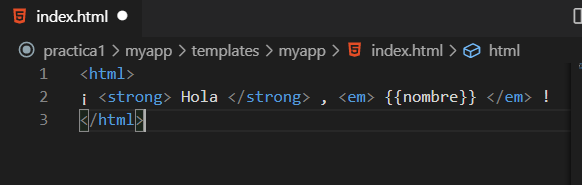
**Contexto y Variables**

Desde una **plantilla** podemos acceder a **variables** definidas en código Python. Para ello, las **variables** se incluyen en un diccionario que se pasa a la función **render()** y al cual se lo conoce como **contexto.**

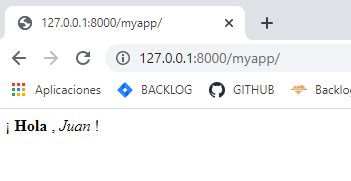
1.Modificamos **views/index.html**:



2. En la plantilla **index.html:**



3. Resultado:



El **valor** de la **key** nombre del **diccionario** correspondiente aparece de esta forma en la plantilla **html**.

4. Hacemos un **nuevo Ejemplo**. Agregamos lo siguiente a la funcion **index()** de views.py:

def index ( request ):

    ctx = {"nombre": "Juan",

    "cursos": 5,

    "curso\_actual": "Python & Django"

    }

    return render(request, "myapp/index.html", ctx)

5. Plantilla:

<html>

    <head></head>

    <body>

        ¡ <strong> Hola </strong> , <em> {{nombre}} </em> !

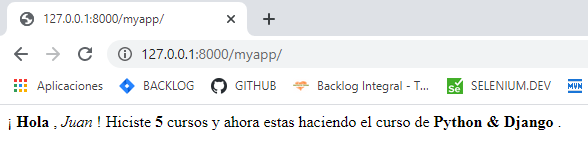
        Hiciste <strong> {{cursos}} </strong> cursos y ahora

        estas haciendo el curso de <strong>{{curso\_actual}}</strong>

    </body>

</html>

6. Resultado:



7. A su vez, si queremos **acceder** a una **colección** dentro del diccionario, lo hacemos mediante el punto.

Actualizamos **index()**

def index ( request ):

    ctx = {

    "nombre": "Juan",

    "cursos": 5,

    "curso\_actual": {"nombre" :"Python & Django", "turno": "Noche"}, #diccionario

    "cursos\_anteriores": ["Java", "PHP", "JavaScript", "Python"] #listas

    }

    return render(request, "myapp/index.html", ctx)

8. **Plantilla index.html**

<html>

    <head>

        <title>"Cursos"</title>

    </head>

    <body>

        ¡ <strong> Hola </strong> , <em> {{nombre}} </em> ! <br>

        Hiciste <strong> {{cursos}} </strong> cursos y ahora

        estas haciendo el curso de <strong>{{curso\_actual.nombre}}</strong>

        en el turno <strong>{{curso\_actual.turno}}. </strong> <br>

        Y tu primer curso fue <strong>{{cursos\_anteriores.0}}.</strong>

    </body>

</html>

9. **Resultado:**



Cabe aclarar que el . “punto” no es código Python, sino mas bien forma parte del sistema de plantillas de Django.

**CONDICIONALES**

1.Funcion **index()** en views.py:

def index ( request ):

    ctx = {

        "nombre": "Juan",

        "cursos": 0

    }

    return render(request, "myapp/index.html", ctx)

2. **Plantilla**:

<html>

    <head>

        <title>"Cursos"</title>

    </head>

    <body>

        ¡<strong>Hola</strong>, <em>{{nombre}}</em>!

        {% if cursos > 0 %}

            hiciste {{cursos}} cursos.

        {% else %}

            no hiciste ningun curso todavia.

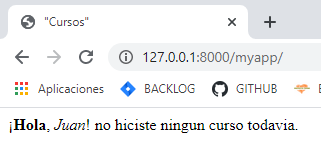
        {% endif %}

    </body>

</html>

Probamos aumentando la cantidad de cursos.

3. Resultado:



**BUCLES**

1.Es el equivalente al FOR de Python. Ideal para construir **listas** o **tablas** HTML a partir de una **colección**.

2. Funcion **index()**:

def index ( request ):

    ctx = {

        "alumnos": ["Juan", "Sofia", "Matias"]

    }

    return render(request, "myapp/index.html", ctx)

3. **Plantilla**:

<html>

    <head>

        <title>"Alumnos"</title>

        <h1>Lista de Alumnos</h1>

    </head>

    <body>

        <ul>{% for alumno in alumnos %}

            <li>{{alumno}}</li>

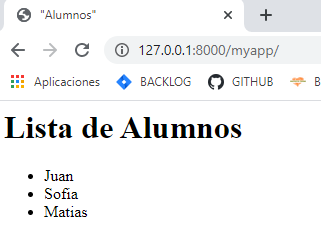
            {% endfor %}

        </ul>

    </body>

</html>

4. **Resultado**:



5. Creamos **views/cursos.py**:

def cursos (request):

    conn = sqlite3.connect("db.sqlite3")

    cursor = conn.cursor()

    cursor.execute("SELECT nombre, inscriptos FROM cursos")

    cursos = cursor.fetchall()

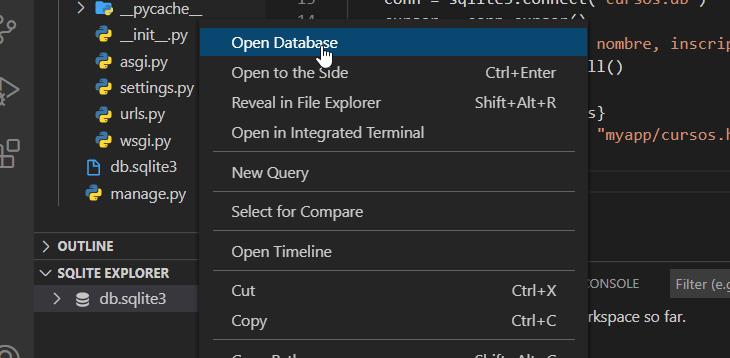
    conn.close()

    ctx = {"cursos": cursos}

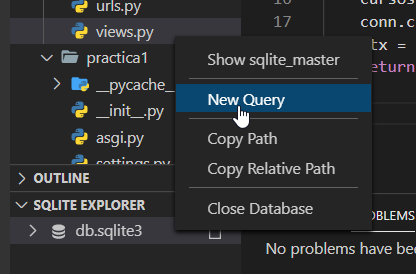
    return render(request, "myapp/cursos.html", ctx)

**IMPORTANTE:** recordar importar el paquete de sqlite3

6. Abrimos la **BBDD** **SQLITE3**:



7. **New Query:**



8. Creamos la tabla **cursos**:

CREATE TABLE "cursos" (

    "id" serial NOT NULL PRIMARY KEY,

    "nombre" varchar(200) NOT NULL,

    "inscriptos" integer NOT NULL

    );

9. Insertamos algunos **valores**:

INSERT INTO cursos (id, nombre, inscriptos)

 VALUES (1, "Jose", 2);

 INSERT INTO cursos (id, nombre, inscriptos)

 VALUES (2, "Maria", 5);

 INSERT INTO cursos (id, nombre, inscriptos)

 VALUES (3, "Juana", 8);

select \* from cursos;

10. **templates/myapp/cursos.html**:

<html>

    <title> Lista de cursos </title>

    <table style = "border: 1px solid" >

        <thead>

            <tr>

                <th> Curso </th>

                <th> Inscriptos </th>

            </tr>

        </thead>

        {% for nombre, inscriptos in cursos %}

            <tr>

                <td> {{ nombre }} </td>

                <td> {{ inscriptos }} </td>

            </tr>

        {% endfor %}

    </table>

</html>

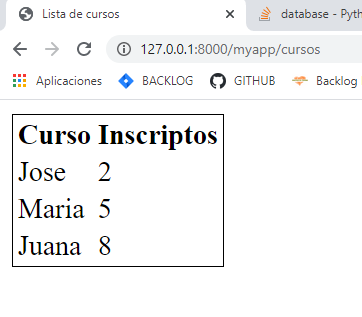
11. **myapp/urls.py**:

Agregamos la línea:

path("cursos", views.cursos, name="cursos"),

a **urlpatterns**.

12. Resultado:



13. Codigo (**CheckPoint**):



**SINTAXIS url, TEMPLATE TAG {% url %}**

1.En **urls.py** de nuestra app cada una de las direcciones URL, la identificamos con un nombre (3er argumento de la función **path()**.

Ej: **urls.py**

urlpatterns = [

path( "" , views.index, name = "index" ),

path("cursos", views.cursos, name="cursos"),

]

Este nombre nos permite generar direcciones URL en las plantillas usando la sintaxis **{ % url … %}** sin necesidad de utilizar el **href=…**.

Es decir este tag nos permite hacer referencia directamente a una view desde nuestros templates (html) y es la forma correcta de escribir enlaces relativos dentro de nuestra web.

2. Por ejemplo:

<a href = '{% url "cursos" %}' > Ver cursos </a>

Genera esto:

<a href = '/myapp/cursos' > Ver cursos </a>

3. El **beneficio** de usar la etiqueta **url** en lugar del **href,** es que podremos modificar la dirección de URL en cualquier momento sin necesidad de modificar las plantillas donde se haga referencia ella.

4. De forma tal que **cambiamos** esto:

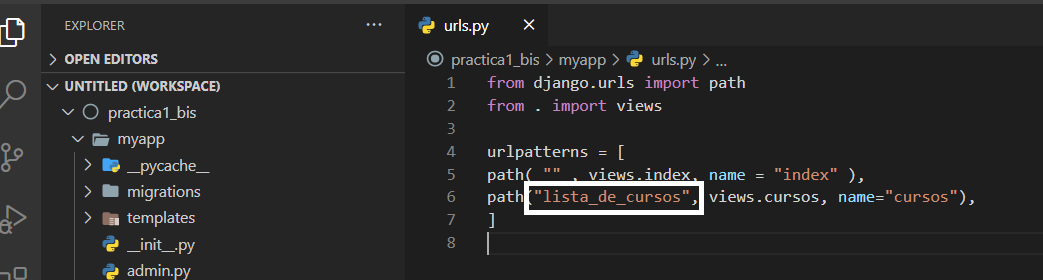
path( "cursos" , views.cursos, name = "cursos" ),

por esto:

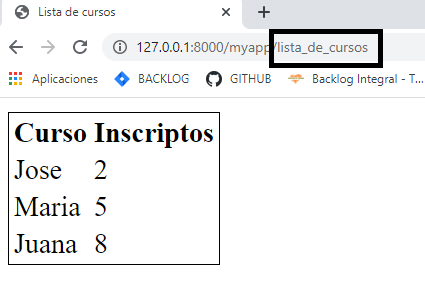
path( "lista-de-cursos" , views.cursos, name = "cursos" ),

el enlace que escribimos anteriormente sigue funcionando, puesto que fue generado en tiempo de ejecución a partir del identificador de la dirección URL (“cursos”).

5. Entonces **cambiamos** el nombre previo **cursos** en **urls.py** por:



6. Refrescamos el browser y obtenemos el mismo resultado:



**ARCHIVOS ESTATICOS** (Pag 8)

1.**Archivos estáticos** son todos aquellos que no sean generados en tiempo real en cada petición (ej: **Javascript, css, imagen, documentos pdf**). Existen varias **razones** para que este tipo de archivos tengan un **trato especial**. Es decir, una vez en producción nuestro proyecto, querremos que los archivos estáticos sean servidos directamente por el **servidor web** en lugar del framework. Para ello debemo usar una **etiqueta especial** en la que hacemos referencia a alguno de ellos.

2. Ubicamos los **archivos estáticos** en la carpeta **myapp/static/myapp/.**

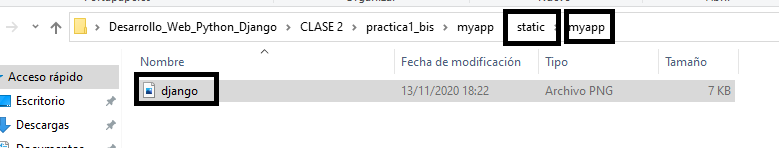
3. El archivo **django.png** a continuación debemos ubicarlo en **myapp/static/myapp/django.png**

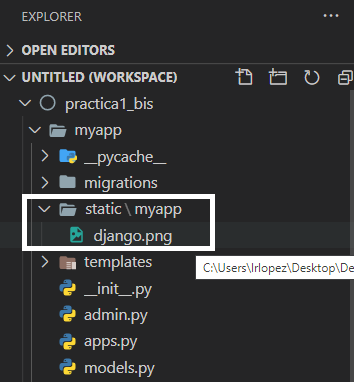
Para eso 1ro creamos la carpeta **static** en myapp y dentro de ella creamos la carpeta **myapp.**

Dentro de la cual vamos a alojar la imagen:



4. Nos queda de la siguiente forma:





5. Ahora para mostrarlo en la plantilla actualizamos **index.html:**

{% load static %} # indica q usaremos la etiqueta static la cual

# no es de Django, sino por una aplicación que

# viene instalada por defecto

<html>

    <head>

        <title>"Alumnos"</title>

        <h1>Lista de Alumnos</h1>

    </head>

    <body>

#usamos la etiqueta o template tag static para generar una

#dirección

#url hacia el archivo static/myapp/django.png

        <img src='{% static "myapp/django.png" %}'>

        <ul>{% for alumno in alumnos %}

            <li>{{alumno}}</li>

            {% endfor %}

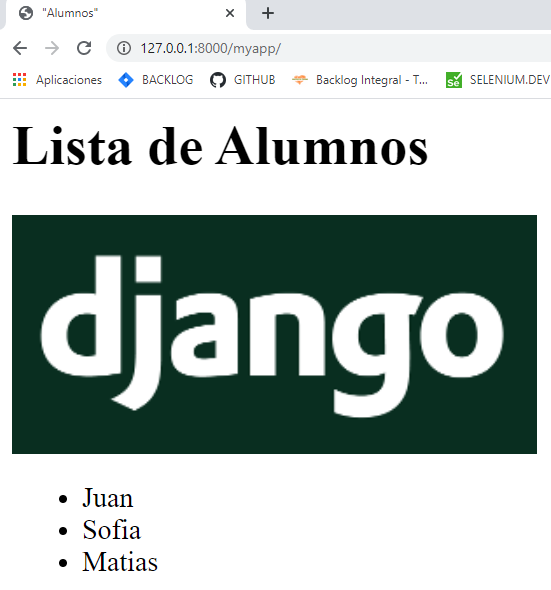
        </ul>

    </body>

</html>

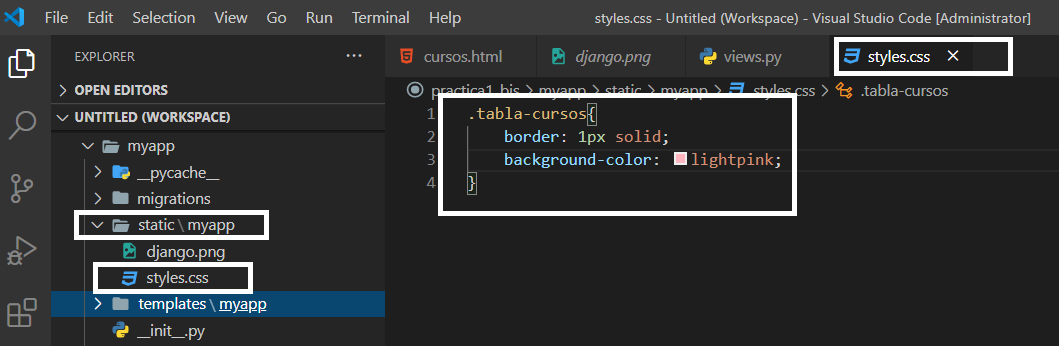
python manage.py runserver **(reiniciamos nuevamente el servidor)**

6. Resultado:



**CSS Y JAVASCRIPT**

1.Creamos un nuevo archivo css templates en **/static/myapp/styles.css:**



2. Cargamos **styles.css** a cursos.html. A través de la etiqueta **link** de html y **static** de Django.

**cursos.html**

{% load static %}

<html>

    <head>

        <title> Lista de cursos </title>

        <link rel="stylesheet"  type="text/css" href='{% static "myapp/styles.css" %}'>

    </head>

    <body>

        <table class="tabla-cursos">

            <thead>

                <tr>

                    <th> Nombre </th>

                    <th> Cursos Inscriptos </th>

                </tr>

            </thead>

            {% for nombre, inscriptos in cursos %}

                <tr>

                    <td> {{ nombre }} </td>

                    <td align="center"> {{ inscriptos }} </td>

                </tr>

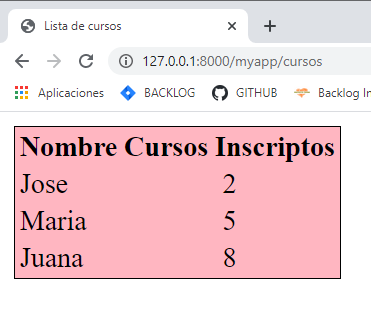
            {% endfor %}

        </table>

    </body>

</html>

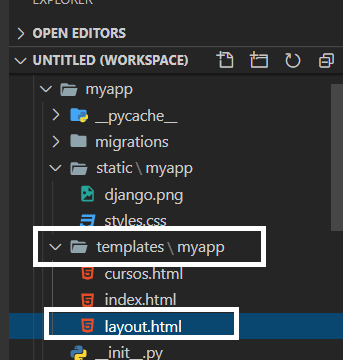
3. Resultado:



**EXTENSION (Pag 10)**

Cuando en las plantillas se da código repetido, es decir varias paginas web tienen los mismos header y footer por ejemplo. Django nos provee la funcionalidad de **herencia de plantillas**.

1.Creamos una plantilla base **myapp/layout.html**:



<!doctype html >

<html>

    <head>

        <title>

        {% block title %}

        Mi aplicación de Django

        {% endblock %}

        </title>

    </head>

    <body>

        <main>

            {% block content %}

            {% endblock %}

        </main>

        <footer>

            <small> &copy; Mi aplicación de Django </small>

        </footer>

    </body>

</html>

Definimos dos bloques via etiqueta de django { % block… % }. Esto nos permitirá luego reemplazar esos valores desde plantillas que extiendan este documento.

2. Por ejemplo en **myapp/index.html** colocamos lo siguiente:

{% extends "myapp/layout.html" %} <!--hereda de layout.html -->

{% load static %}

<html>

    <head>

       <!-- Este bloque reemplazará el título original "Mi aplicación de Django"

Que viene del archivo padre o template base. -->

        {% block title %}

            Lista de alumnos

        {% endblock %}

    </head>

    <body>

      <!-- Este bloque se ubicará dentro de la etiqueta <main> de layout.html. -->

        {% block content %}

        <h1> Lista de alumnos </h1>

            <ul>

                {% for alumno in alumnos %}

                    <li> {{ alumno }} </li>

                {% endfor %}

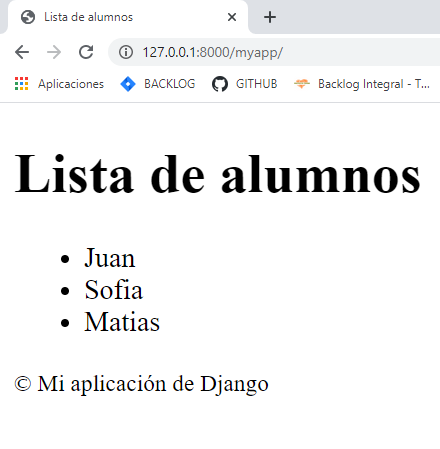
            </ul>

        {% endblock %}

    </body>

</html>

3. Resultado:



En este caso el header y el footer se pueden utilizar en distintas plantillas. Por ejemplo que pasaría si le sacamos este código a **index.html**:

   {% block title %}

           Lista de alumnos

        {% endblock %}

Vemos que muestra el titulo del template base.

4. Codigo Checkpoint:

****

**CAPTURAR VALORES EN DIRECCIONES URL**

Previamente hemos creado la dirección de URL /cursos y la asociamos con la vista cursos() en **urls.py:**

path("cursos", views.cursos, name="cursos"),

donde se mostraba una lista de cursos guardados en la tabla **cursos** de la base **db.sqlite3**.

1.Creamos otra tabla llamada **cursos\_listado**:

-- SQLite

CREATE TABLE IF NOT EXISTS cursos\_listado (

    id int PRIMARY KEY,

    nombreCurso VARCHAR(200),

    cantInscriptos int DEFAULT 0

);

2. Le cargamos **datos**:

INSERT INTO cursos\_listado (id, nombreCurso, cantInscriptos) VALUES

(1, 'PHP', 10),

(2, 'JAVA', 10),

(3, 'Python', 10),

(4, 'Desarrollo Web Python Django', 10);

3. Supongamos ahora que queremos crear una vista curso() donde se muestre el detalle de un curso en particular. Para ello podríamos configurar una dirección de URL /curso a la que suceda el nombre del curso del cual se quiere retornar la información, por ejemplo:

/curso/PHP

/curso/JAVA

/curso/Python

/curso/Desarrollo Web Python Django

Entonces **añadimos** esa opción a nuestro **myapp/urls.py:**

path("curso/<str:nombre\_curso>", views.curso, name="curso"),

de forma tal que el código nos queda:

from django.urls import path

from . import views

urlpatterns = [

path( "" , views.index, name = "index" ),

path("cursos", views.cursos, name="cursos"),

path("curso/<str:nombre\_curso>", views.curso, name="curso"),

]

Crearemos una nueva vista (función) llamada **curso** y donde la novedad <str:nombre\_curso> en la dirección de **URL** Le indica a Django que tiene que “capturar” una cadena que estará ubicada en ese lugar en la **URL** y que deberá **guardarla** en la variable *nombre\_curso.* Luego será pasada como argumento a la vista **curso().**

4. La vista **curso()** quedara de la siguiente manera:

def curso(request, nombre\_curso):

    conn = sqlite3.connect("db.sqlite3")

    cursor = conn.cursor()

    cursor.execute("SELECT nombreCurso, cantInscriptos FROM cursos\_listado WHERE nombreCurso=?",

                   [nombre\_curso])

    curso = cursor.fetchone() #clase cursor, metodo fetchone() de la clase cursor

    ctx = {"curso": curso}  #creamos un diccionario

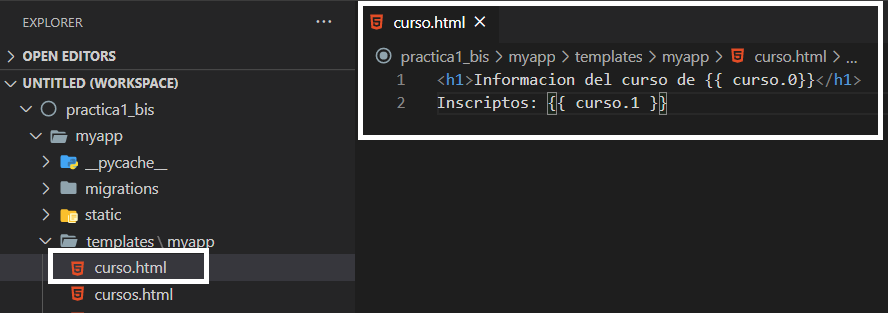
    conn.close()

    return render(request, "myapp/curso.html", ctx)

Usamos el argumento nombre\_curso (en el cual Django puso el nombre del curso que “capturo” de la dirección de URL) para obtener la información de ese curso en particular a partir de una consulta SQL (where nombre=?).

Entonces si el usuario ingresa <http://127.0.0.1:8000/myapp/curso/Python>, entonces el valor de **nombre\_curso** es “Python”, si ingresa <http://127.0.0.1:8000/myapp/curso/JAVA>, entonces **nombre\_curso** es “Java” y asi sucesivamente.

5. Creamos la plantilla **templates/myapp/curso.html:**



De esta forma el **nombreCurso** se va a ubicar en **curso.0** y la **cantInscriptos** se va a ubicar en **curso.1**.

6. **Resultado**:



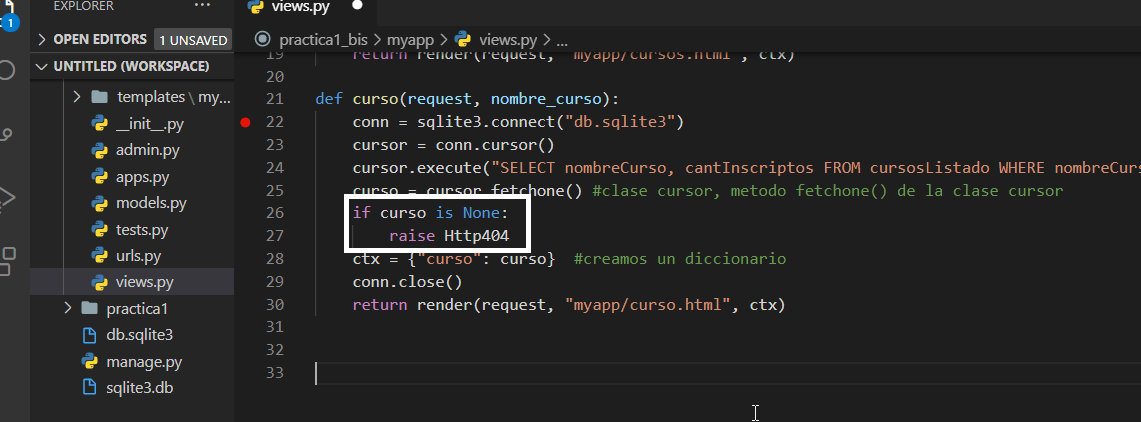
7. **Codigo:**



8. **EXCEPCIONES**:

Si **cursor.fetchone()** retorna **None** quiere decir que el nombre del curso que tenemos guardado en **nombre\_curso** no se corresponde con alguno de la base de datos. En este caso podemos **generar** un **error 404 (Not Found)** lanzando la excepción *django.http.Http404:*

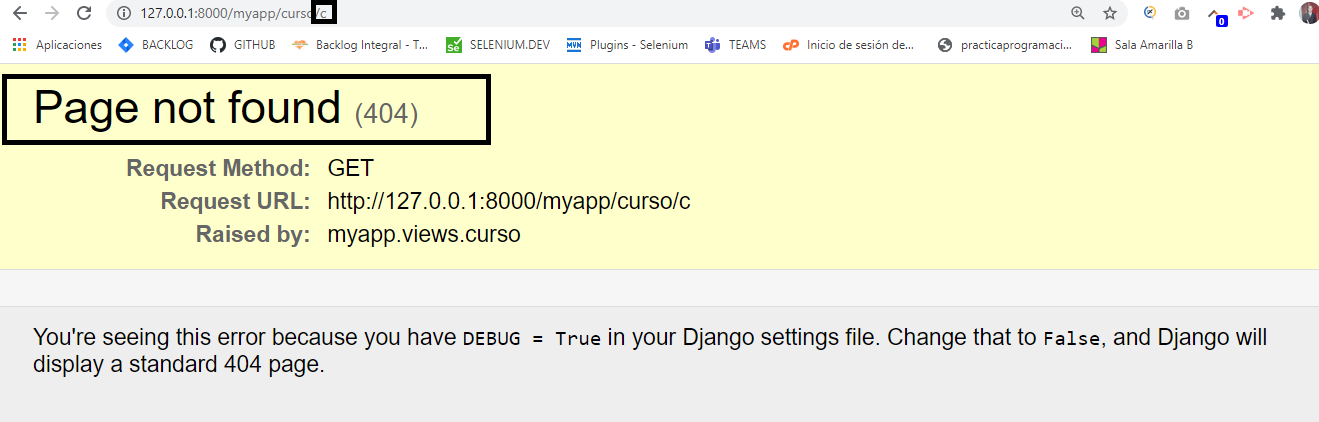
Para ello en **views.py** agregamos el siguiente código:



E importamos el paquete:

from django.http import Http404

9. **Resultado:**

****

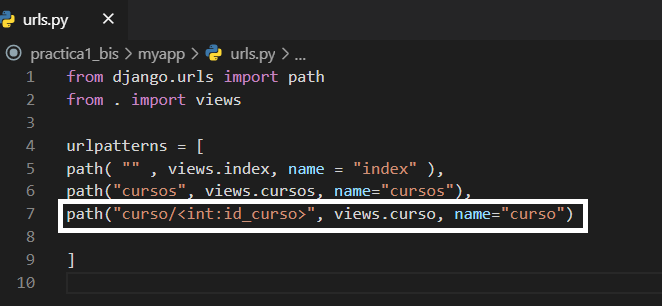
Si comentamos la estructura **IF** agregada previamente en **views.py**:



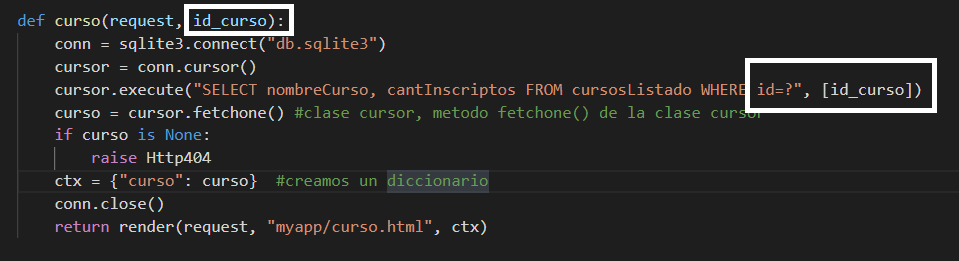
No dándose el error 404 deseado.

**CAPTURAR ENTEROS desde la URL:**

1.Si cada curso tuviese un identificador numerico, podríamos hacer en **myapp/urls.py:**



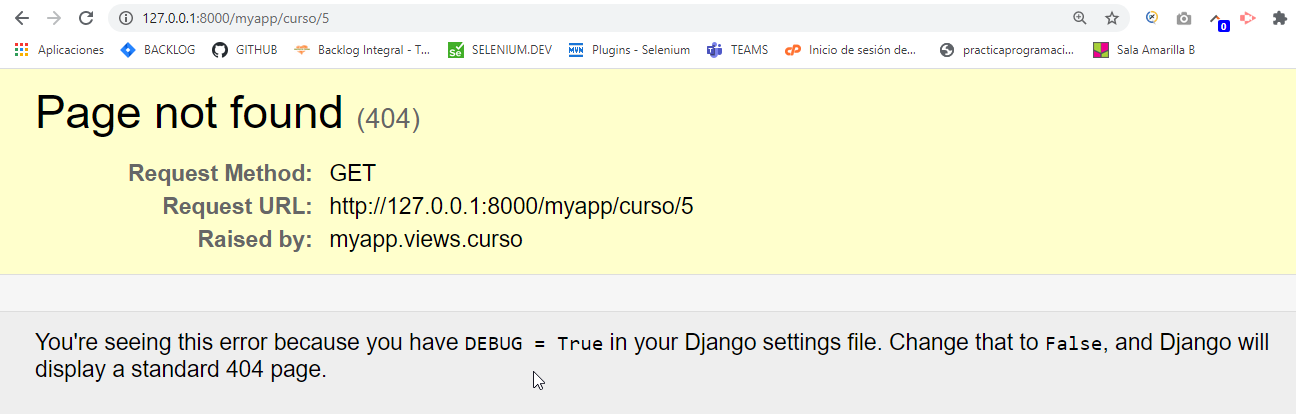
2. **views.py:**

****

3. **Resultado**:



4. Y en caso de poner un nro que **no** exista en tabla (Error 404):



**LABORATORIOS**



**EJERCICIO\_1**

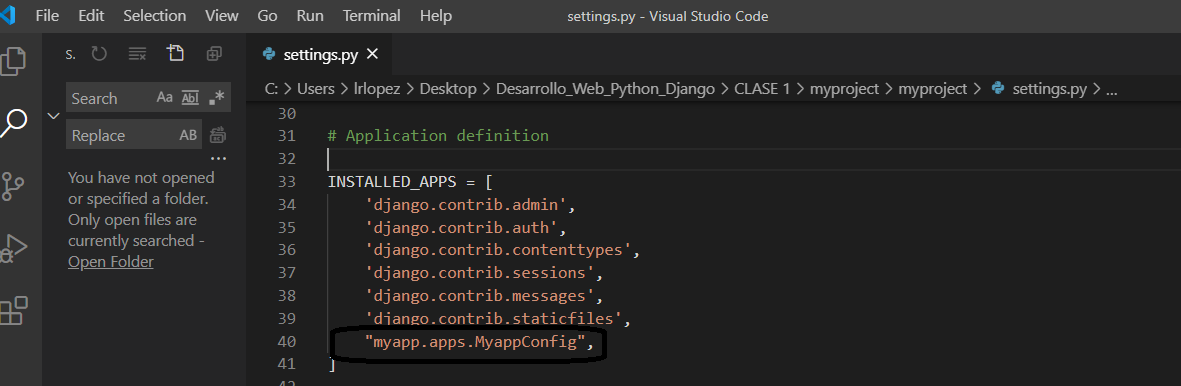
1.Creamos el proyecto **ejercicio1**:

django-admin startproject ejercicio1

2. Creamos una nueva **aplicación** dentro del directorio **ejercicio1** (donde se encuentra el **manage.py**):

python manage.py startapp myapp

4. Asociamos la app creada **myapp** a nuestro proyecto:



5. Migramos las modificaciones:

python manage.py migrate

6. Iniciamos el servidor:

python manage.py runserver

7. Ruta:

C:\Users\lrlopez\Desktop\Desarrollo\_Web\_Python\_Django\CLASE 2\Laboratorio\ejercicio1>

8. **urls.py (myapp)**:

from django.urls import path

from . import views

urlpatterns = [

   path("peliculas/<str:nombrePelicula>/comentarios/<int:nroComentario>", views.peliculas, name="peliculas"),

]

9. **views.py**:

import requests

from django.http import HttpResponse

def peliculas(request, nombrePelicula, nroComentario):

    return HttpResponse(

        f"Comentario numero {nroComentario} de la pelicula {nombrePelicula}." #donde f{} representa el manejo de los f-strings

    )

**Urls.py** (proyecto): agregamos la linea

    path("myapp/", include("myapp.urls"))

de forma tal que nos queda:

from django.contrib import admin

from django.urls import path, include

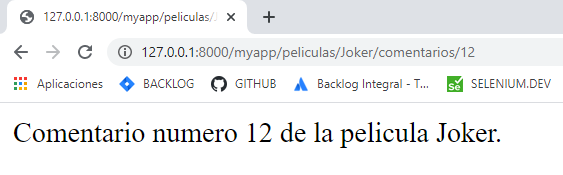
urlpatterns = [

    path('admin/', admin.site.urls),

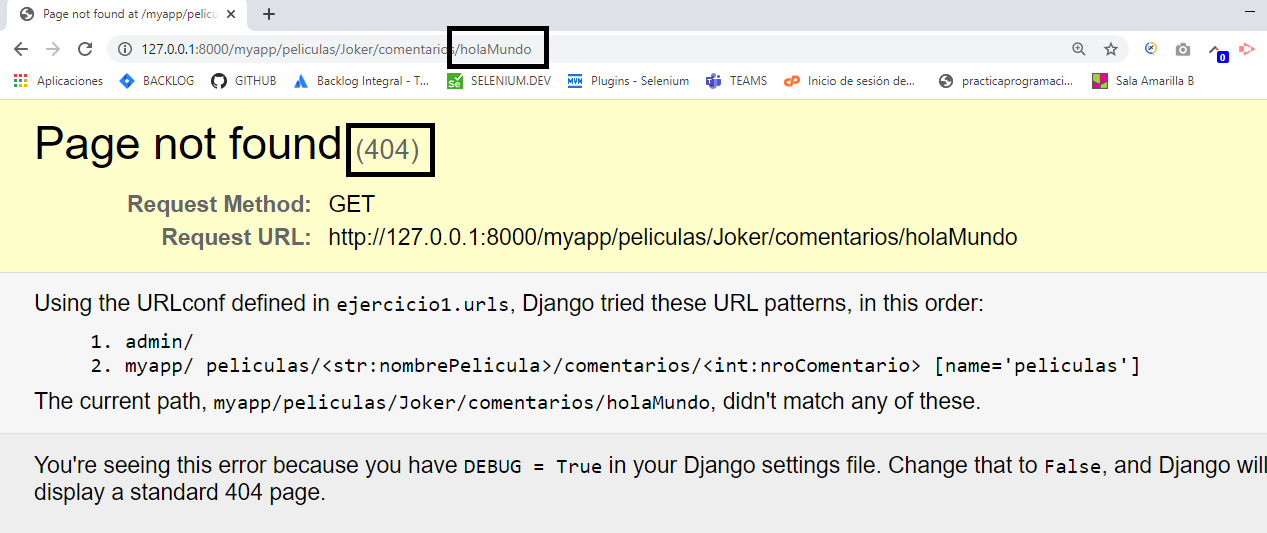
    path("myapp/", include("myapp.urls"))

]

10. **Resultado:** En este caso no fue necesario crear un html o una tabla. Simplemente usamos los datos ingresados via url.



11. Si en lugar de un entero colocamos un String obtenemos un error **404:**

****

Esto se da porque estamos ingresando un tipo de dato diferente.

12. **Codigo:**

****

**EJERCICIO\_2**

1.Tomamos como base el ejercicio 2 de la clase 1

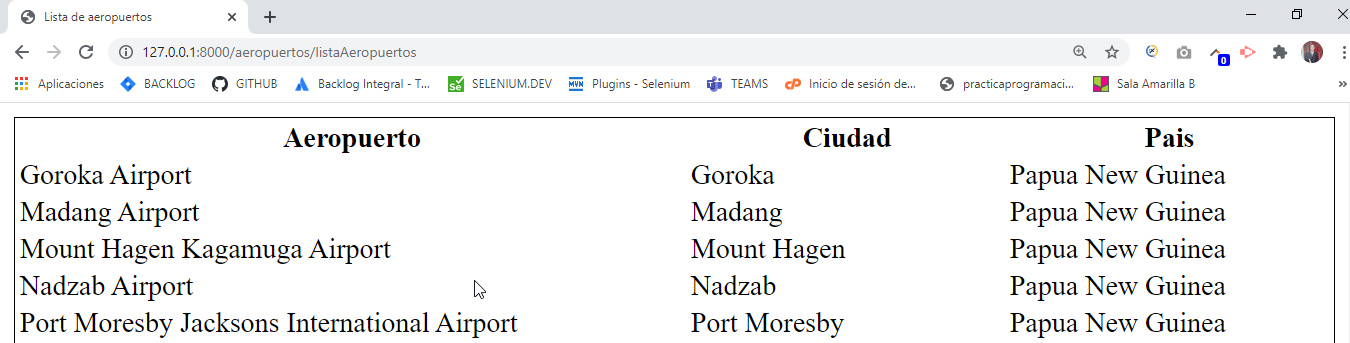


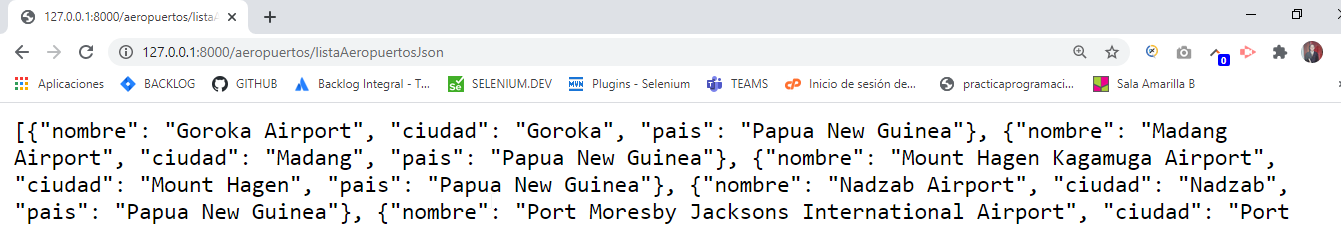
2. Arrancamos el servidor:

C:\Users\lrlopez\Desktop\Desarrollo\_Web\_Python\_Django\CLASE 2\Laboratorio\ejercicio2>

python manage.py runserver

3. En donde teníamos las siguientes salidas





4. Mostramos nuevamente la 1er salida 127.0.0.1:8000/aeropuertos/listaAeropuertos , pero usando las plantillas y generando las filas de la tabla via {% for …%}

5. Agregamos la siguiente línea a urlpatterns en **urls.py**:

    path("listaAeropuertosTabla", views.listaAeropuertosTabla, name="listaAeropuertosTabla")

6. Agregamos en **views.py:**

**Importamos**

from django.shortcuts import render

Y agregamos la siguiente función:

def aeropuertosTabla(request):

    f = open("aeropuertos.csv", encoding="utf8")

    lista\_aeropuertos = []

    for linea in f:

        datos = linea.split(",")

        nombre = datos[1].replace('"', "")

        ciudad = datos[2].replace('"', "")

        pais = datos[3].replace('"', "")

        lista\_aeropuertos.append((nombre, ciudad, pais)) #agrega una sublista a la lista\_aeropuertos

    f.close()

    ctx = {"lista\_aeropuertos": lista\_aeropuertos} #agrega la lista al diccionario

    return render(request, "aeropuertos/aeropuertos.html", ctx)

7. **template/aeropuertos** agregamos **aeropuertos.html**

<html>

  <title>Lista de aeropuertos</title>

  <table style="border: 1px solid">

    <thead>

      <tr>

        <th>Aeropuerto</th>

        <th>Ciudad</th>

        <th>País</th>

      </tr>

    </thead>

    {% for nombre, ciudad, pais in lista\_aeropuertos %}

      <tr>

        <td>{{ nombre }}</td>

        <td>{{ ciudad }}</td>

        <td>{{ pais }}</td>

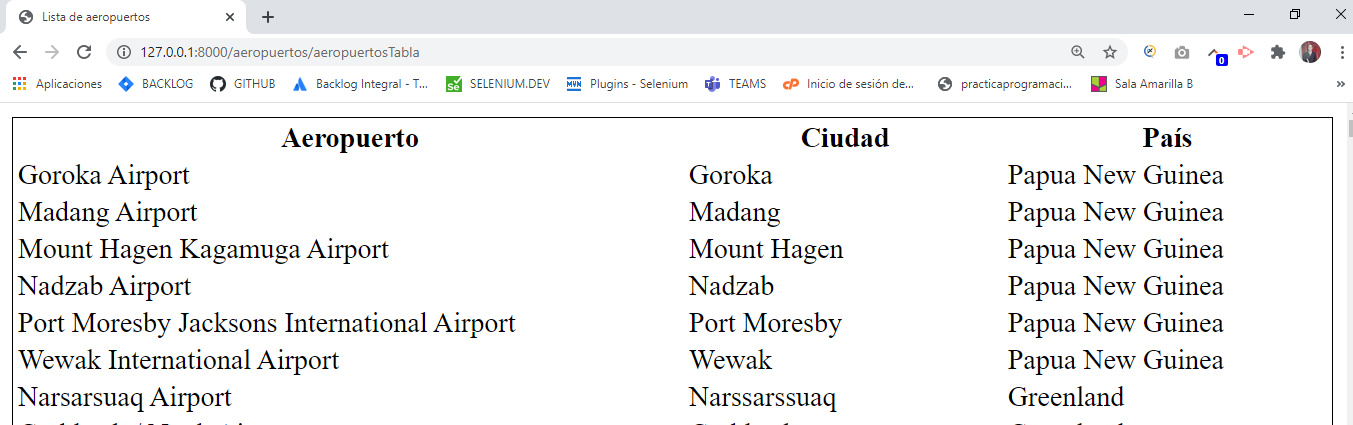
      </tr>

    {% endfor %}

  </table>

</html>

8. Resultado:



9. **Codigo Resultante:**

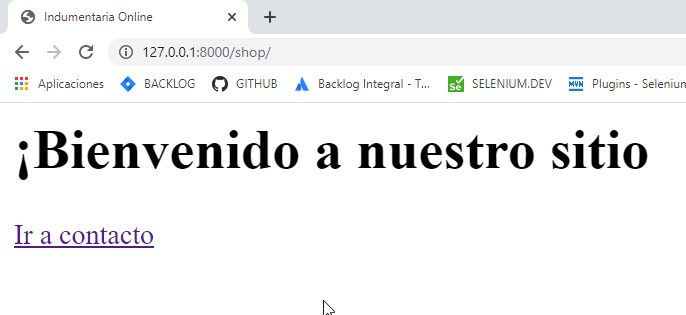


**PROYECTO INTEGRADOR**

1.Partimos del siguiente código (resultado del integrador clase 1):



2. Y llegábamos a

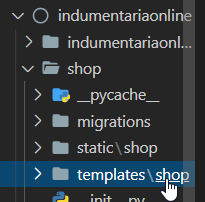


Y a



3. En la clase 1 vimos como generar directamente el código HTML desde las vistas **index()** y **contacto()**. Para continuar avanzando y en base al nuevo tema crearemos **plantillas** para ambas vistas. Y también crearemos un archivo **base.html** a partir del cual hereden las plantillas.

4. Creamos la carpeta **templates**. Y dentro creamos **shop**. De forma tal que nos queda:



5. **index.html** (shop/templates/shop)**:**

En donde generamos las direcciones URL de contacto y shop via etiqueta { %url% } y extendemos de un archivo base.html el cual no creamos aun.

**Index.html**

{% extends "shop/base.html" %}

{% block title %}Pagina Inicial {%endblock %}

{% block content %}

<h1>¡Bienvenido a nuestro sitio!</h1>

<p>

    <a href="{% url 'contacto' %}">Ir a contacto</a>

</p>

{%endblock%}

**6.** Creamos **contacto.html**

{% extends "shop/base.html" %}

{% block title %} Contacto {% endblock%}

{% block content %}

<h1>Contacto</h1>

<p><strong>Direccion</strong>: Av. Le Vayer 1234</p>

<p><strong>Telefono</strong>: +541157614322</p>

<p><strong>Email</strong>: contacto@indumentaria-online.com</p>

<p><a href="{% url 'index'%}">Volver al inicio</a></p>

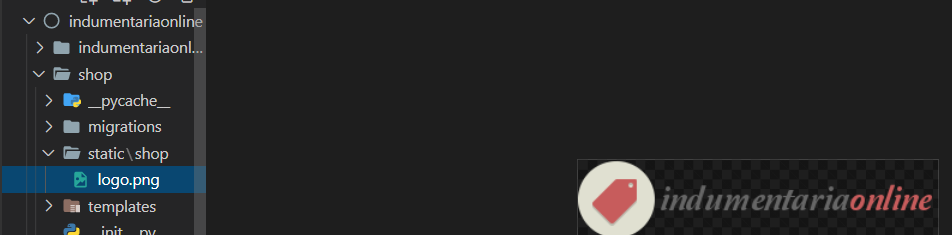
{% endblock %}

7. Creamos el template padre, **base.html.** Alli colocamosel logo del sitio (imagen) usando { %static …% }. Y con un enlace a la pagina de inicio. Para ello:

1. Creamos las carpetas **static / shop** dentro de **indumentariaonline** y le agregamos la imagen logo.png siguiente:



De forma tal que nos quede:



1. Y ahora si creamos el **base.html,** con un enlace a **index.html:**

Y tambien le agregamos blocks en el titulo y en el body también, de forma tal que nos queda:

{% load static %}

<html>

<head>

    <title>

        {% block title %} {% endblock %}

    </title>

</head>

<body>

<a href="{% url 'index' %}">

    <img src="{% static 'shop/logo.png' %}" >

</a>

   {% block content %}{% endblock %}  <!--generamos un bloque lo que significa que en las

                                        plantillas que extienden o heredan de base.html

                                         se aplicara codigo html donde se marque el bloque

                                         -->

</body>

</html>

8. Hacemos los cambios pertinentes en **views.py** (recordar importar render):

from django.shortcuts import render

def index(request):

   return render(request, "shop/index.html")

def contacto(request):

   return render(request, "shop/contacto.html")

Observemos que todo el html que figuraba creado antes, pasa a estar representado en el html correspondiente dentro de **templates/shop**.

**Contexto:** un contexto es un conjunto de variables y sus valores asociados. Una plantilla usa las variables para llenar y evaluar las etiquetas de bloque. **Render()** lo que hace es tomar el contexto para “llenar” la plantilla.

***Ej con el Shell de Python:***

python manage.py Shell

In [1]: from django.template import Context, Template

In [2]: t = Template("Mi nombre es: {{nombre}}.")

In [3]: c = Context({"nombre": "Leonardo"})

In [4]: t.render(c)

Out[4]: 'Mi nombre es: Leonardo.'

Y chequeamos que el **urls.py** de la aplicación nos quede asi:

from django.urls import path

from . import views

urlpatterns = [

    path("", views.index, name="index"),

    path("contacto", views.contacto, name="contacto")

]

9. Colocamos un **block content** en cada plantilla.

Como se mencionó, es usado para sobreescribir partes especificas del template.

Es importante tener en cuenta que cuando abramos o cerremos las llaves no debe haber espacios entre “%” y “{“. O entre “%” y “}”. De forma tal que quede {% %}.

8. **Resultado:**

Desde **index.html**

****

Seleccionando la **imagen** o **Ir a contacto** nos lleva a **contacto.html**.



Una de los beneficios que tiene Django es la **extension** de **plantillas.** En donde podemos **reutilizar** partes del HTML para diferentes paginas del sitio Web. Las plantillas son útiles cuando quieres utilizar la misma información o el mismo diseño en más de un lugar. De forma tal que no sera necesario cambiar todas las plantillas, sino que con modificar la **base.html** es suficiente.

9. **Codigo**:

