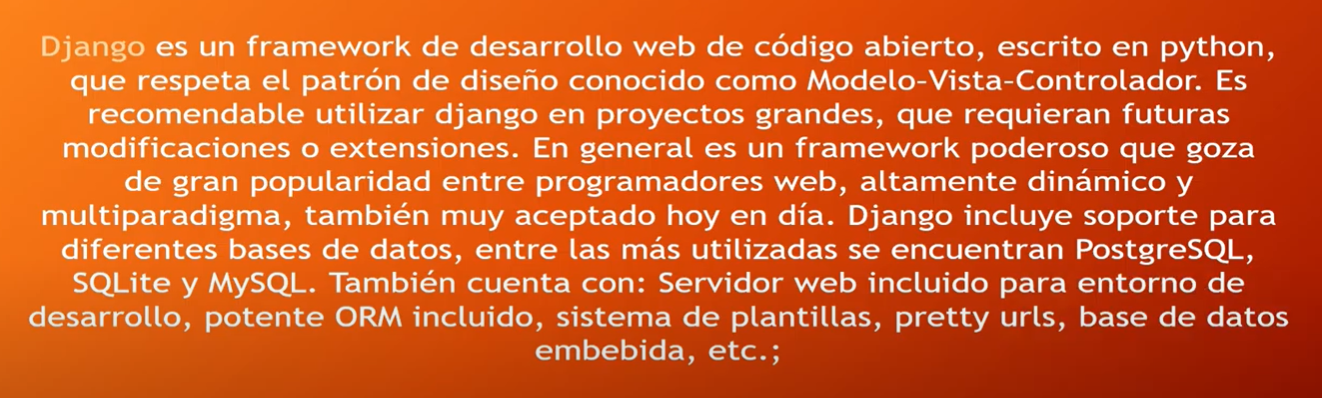
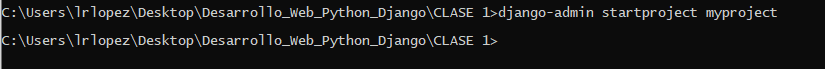
**Introduccion**



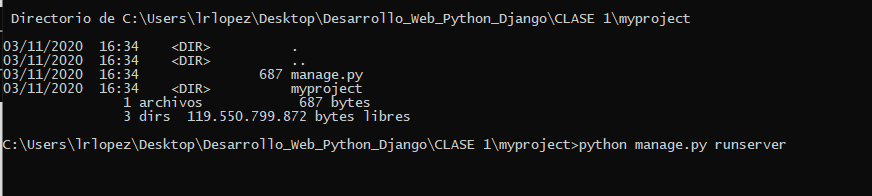
1. Tomamos el Manual del Alumno como referencia:



2. Creamos un Proyecto (pag 8).

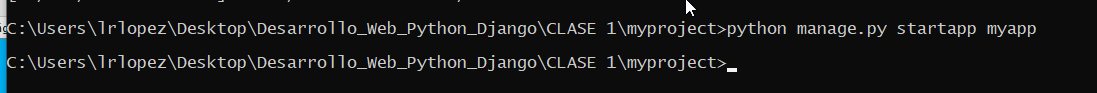


3. Iniciamos el sitio.

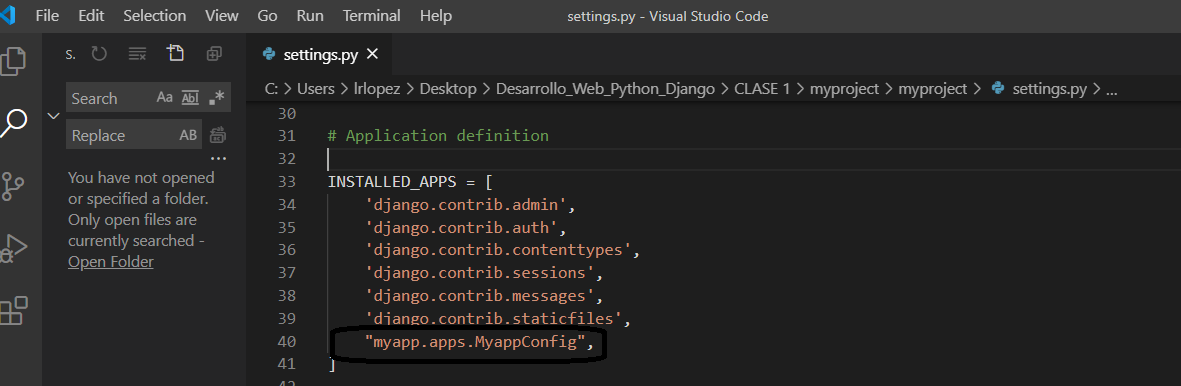


4. Creamos una nueva aplicación (Pag 9):

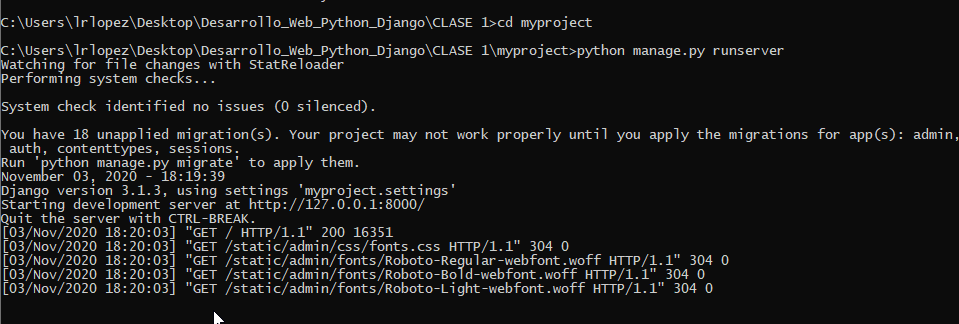
CTRL+C



5. Asociar la app creada myapp a nuestro proyecto:



6. Iniciamos nuevamente el servidor:



**Vistas y Direcciones de URL (Pag 10)**

1. Creamos una vista.

from django.http import HttpResponse

def index(request):

    return HttpResponse("Hola Mundo!")

2. Asociaciones entre URL y Vistas. urls.py

from django.urls import path

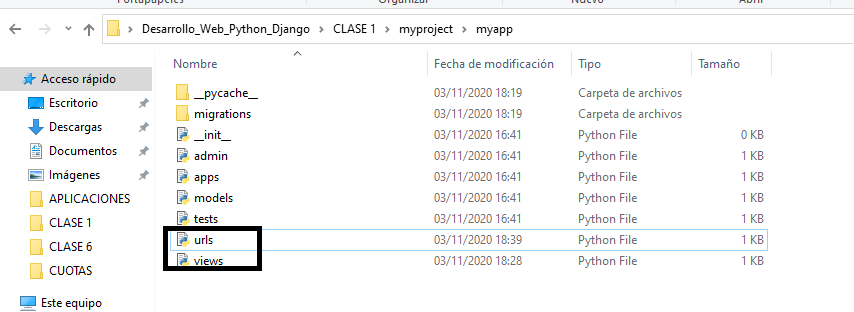
from . import views

urlpatterns = [

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

]

3. Nos queda la carpeta de esta forma:



4. Configuramos **urls.py** del proyecto:

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

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

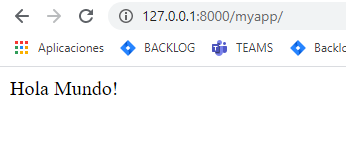
]

Migramos: python manage.py migrate

Iniciamos el servidor: python manage.py runserver

Vamos a : <http://127.0.0.1:8000/myapp/>

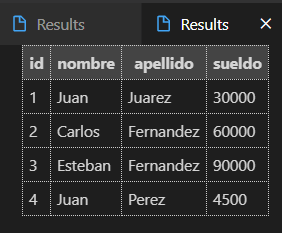
Y llegamos a:

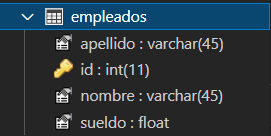


5. Configurar la BBDD MySQL desde Visual Studio (pag 13)



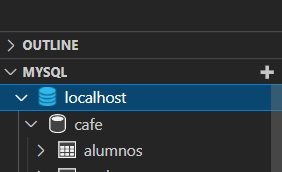
1. Levantamos el esquema **Café** Mysql a Visual studio Code siguiendo el instructivo adjunto.
2. Creamos la tabla **empleados**. Que contenga los siguientes campos:





1. **Settings.py**:

Agregamos **Multiple databases** desde settings.py para indicarle a django que utilizaremos también la base mysql subida recientemente:



DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': 'db.sqlite3',

    },

    'mysqlDDBB': {

        'NAME' : 'cafe',

        'HOST' : '127.0.0.1',

        'ENGINE' : 'django.db.backends.mysql',

        'USER' : 'root',

        'PASSWORD' : '13T-root\_pass\*',

    }

}

**Nota**: esta configuración no es necesaria para contectarnos a la BBDD. Desde **views.py** se puede indicar la conexión.

1. **urls.py** de la aplicacion

from django.urls import path

from . import views

urlpatterns = [

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

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

]

1. Recordamos instalar mysql:

Pip install mysqlclient

1. **Views.py**:

from django.http import HttpResponse

import mysql.connector

def index(request):

    return HttpResponse("Hola Mundo!")

def empleados ( request ):

    conn = mysql.connector.connect( user='root', password='13T-root\_pass\*', host='127.0.0.1', database="cafe", auth\_plugin='mysql\_native\_password' )

    cursor = conn.cursor()

    cursor.execute( "SELECT nombre, apellido, sueldo FROM empleados" )

    html = """

        <html>

            <title>Empleados</title>

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

                <thead>

                <tr>

                <th>Nombre</th>

                <th>Apellido</th>

                <th>Sueldo</th>

                </tr>

            </thead>

            """

    for (nombre, apellido, sueldo) in cursor.fetchall():

        html += f"""

        <tr>

            <td> { nombre } </td>

            <td> { apellido } </td>

            <td> { sueldo } </td>

        </tr>

              """

    html += "</table></html>"

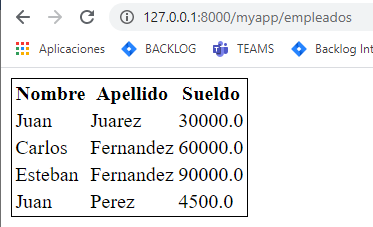
    conn.close()

    return HttpResponse(html)

1. Vamos al sitio correspondiente:

<http://127.0.0.1:8000/myapp/empleados>

Y nos traemos el contenido de la tabla **empleados** Mysql.



Checkpoint codigo:



**Retorno en otros formatos** (Pag 14)

1.Agregamos la siguiente función al archivo views.py

Recordar importar el paquete JsonResponse

from django.http import HttpResponse, JsonResponse

Luego:

def productosJson(request):

    conn = mysql.connector.connect(user='root', password='13T-root\_pass\*', host='127.0.0.1', database="cafe", auth\_plugin='mysql\_native\_password' )

    cursor = conn.cursor()

    cursor.execute("SELECT nombre, apellido, sueldo FROM empleados")

    response = JsonResponse(cursor.fetchall(), safe=False)

    conn.close()

    return response

2. Agregamos la URL a urls.py:

from django.urls import path

from . import views

urlpatterns = [

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

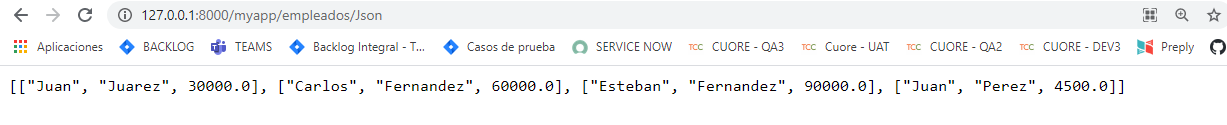
    path("empleados", views.empleados, name="empleados"),

    path("empleados/Json", views.productosJson, name="productosJson"),

]

3. Accedemos al sitio:

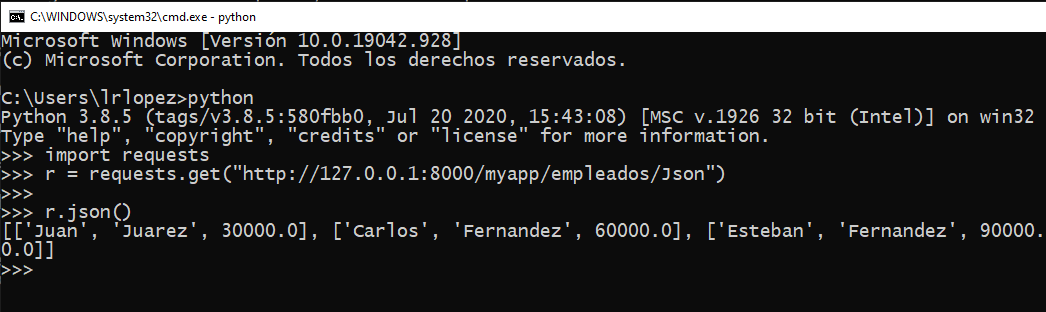
http://127.0.0.1:8000/myapp/empleados/Json



4. Tambien podemos acceder desde la consola interactiva:

Dejamos arriba el servidor corriendo (python manage.py runserver).

Abrimos otro cmd y oprimimos **python**. Nos lleva a la consola interactiva de python. Luego



**Codigo Definitivo:**

****

**Ver versión de Django**

python -m django --version

**Laboratorio 1:**

a. **Creamos nuestro proyecto**:

django-admin startproject lab1

b.  **Arrancamos nuestro proyecto:**

python manage.py runserver

python manage.py runserver 8080 (cuando queremos cambiar el puerto del servidor)

c. <http://127.0.0.1:8000/> para ver andando nuestro proyecto.

d. **Creamos las aplicaciones** (que interactuaran con el proyecto)

ctrl+c

python manage.py startapp ejercicio1 (posicionados sobre el mismo nivel que manage.py)

e. **Indicamos a Django que la app ejercicio1 creada pertenece a nuestro proyecto Laboratorio1\_Profe**

Laboratorio1\_Profe/settings.py

'ejercicio1.apps.Ejercicio1Config' (colocar en settings.py dentro de INSTALLED\_APPS) Se coloca Ejercicio1Config porque es el nombre de la clase que contiene apps.py.

python manage.py runserver (iniciamos de nuevo el servidor)

f. **Asociamos url** y **vista.**

urls.py

from django.urls import path

from . import views

urlpatterns = [

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

]

**Estructura de un Proyecto en Django:**

manage.py: es un intermediario, a partir de este archivo podemos hacer cosas.

\_\_init\_\_.py: es un indicador que le permite a Django reconocer un script de python. Si este archivo no existe, no reconoce los script de Python

**Settings.py:** Secret\_key: es una clave privada por proyecto.

Debug=True: cuando estamos en producción pasa a False

INSTALLED\_APPS: contiene aplicaciones propias que trae Django instaladas.

auth: usuarios, messages: de validación, errores, etc

Aquí van todas las aplicaciones que vayamos creando.

MIDDLEWARE: Comunicación de lo que envia el cliente al servidor y visceversa.

Verifica que no se saquen ciertas jerarquías, permisos y niveles.

ROOT\_URLCONF: Tenemos las rutas principales de nuestro proyecto. Hace

referencia al archivo **urls.py.**

TEMPLATES: Se utiliza para las plantillas. Utiliza un sistema DjangoTemplates

integrado para las plantillas. DIRS: directorios, ruta donde vamos a alojar los

templates. APP\_DIRS: TRUE: cuando el cliente haga una petición de una plantilla

(html) si esta en verdadero incluye la búsqueda en el direcotrio.

WSGI\_APPLICATION: es para producción. Como levanta la aplicación y como

recibe las comunicaciones. Llama a wsgi.py.

DATABASES: POSTGRE, MYSQL, SQLITE3. Por defecto viene configurado para

SQLITE3.

AUTH\_PASSWORD\_VALIDATORS: Son validaciones de passwords para los

usuarios.

**urls.py:** Es el archivo principal de rutas.

path es la vista que tiene incorporadas las rutas en Django.

Urlpatterns: *path(‘aquí va lo que se escribe en el navegador’, aquí va la vista)*

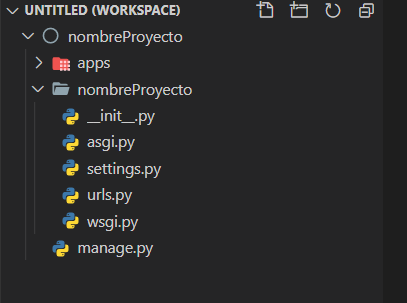
Modelo vista template en Django. Asi es como esta configurado el path.

**wsgi.py:** crea una instancia. E indica la configuración (settings).

**MODELO VISTA TEMPLATE – Creamos una app Biblioteca**

**Aplicacion:** es una carpeta que tiene una serie de archivos que trabajaran en una parte de la lógica o una sección de la BBDD. Si queremos trabajar sobre la administración de usuarios, tendremos otra carpeta para tal efecto.

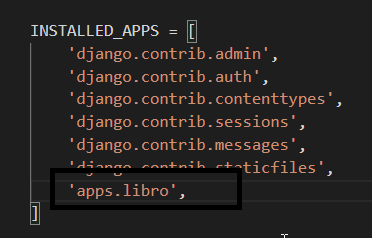
Estructura inicial de un proyecto:



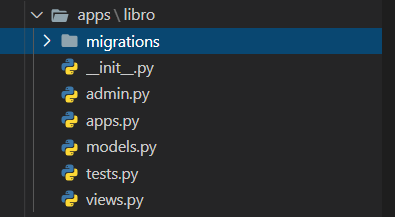
Creamos una carpeta apps para poner dentro todas las aplicaciones.

1.django-admin startapp libro (creamos una aplicación llamada libro dentro de apps)

2. dentro de settings.py del proyecto registramos la app libro.

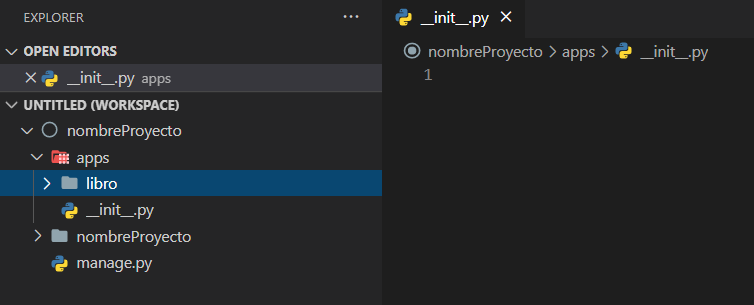


3. Entramos a la carpeta libro



**migrations:** contendrá los archivos que intervengan cuando tengamos que hacer una migración. Por ejemplo cuando tenemos una estructura de datos en nuestra app y queremos mandarla al gesto de base de datos para que se cree una tabla por ejemplo o se carguen datos.

\_\_init\_\_.py: indica que aquí hay archivos de Python (es un archivo de enlace). Si no esta no reconoce los archivos de Python. Creamos un ­­\_\_init\_\_.py a nivel de la carpeta apps:



Lo agregamos para que reconozca archivos de Python

**admin.py:** Registramos modelos y tablas para que sean incluidas al administrador que viene incluido en Django.

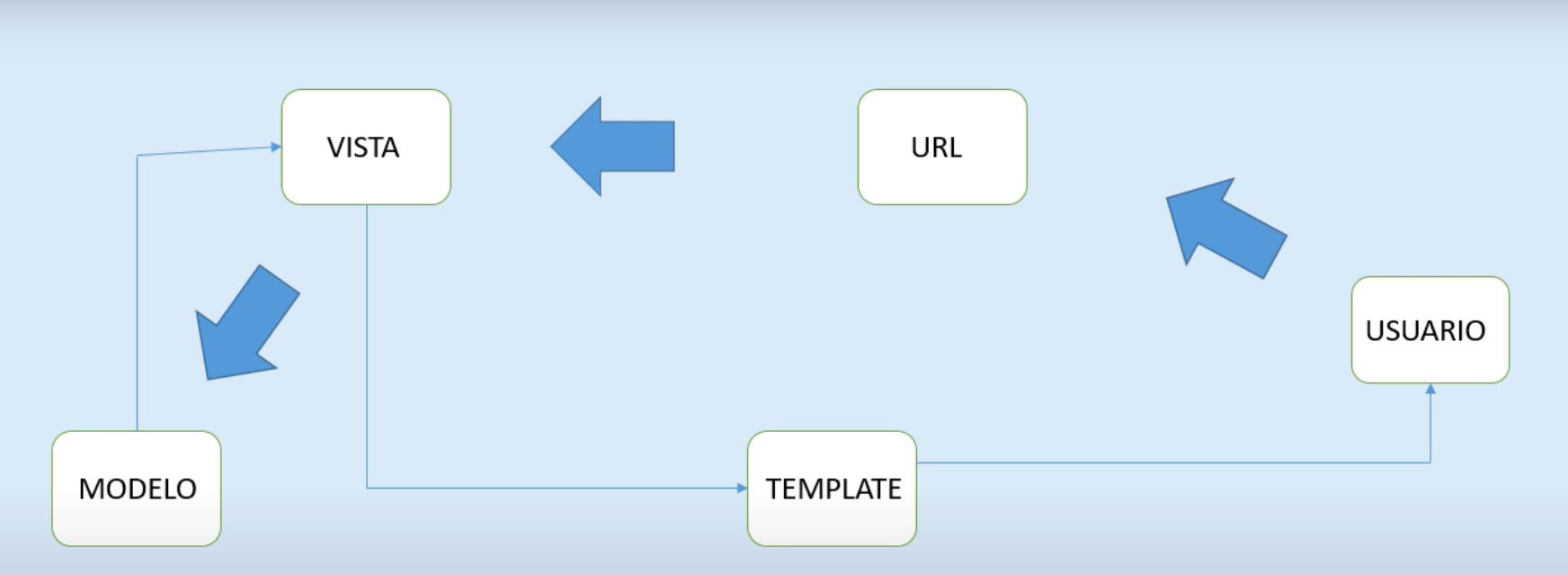
**apps.py:** indica la configuración de una aplicación

**models.py:** tendremos un modelo o clase hecha en Python que va a representar una tabla en la BBDD. Podemos interactuar con la BBDD. Indicando las tablas.

**tests.py:** Permite hacer testeo de nuestra aplicación, de nuestros modelos, o tablas.

**views.py:** van a ir la lógica de nuestra aplicación en donde vamos a tener el código que permite crear archivos relacionados a una aplicación.

**Modelo-Vista-Template:**



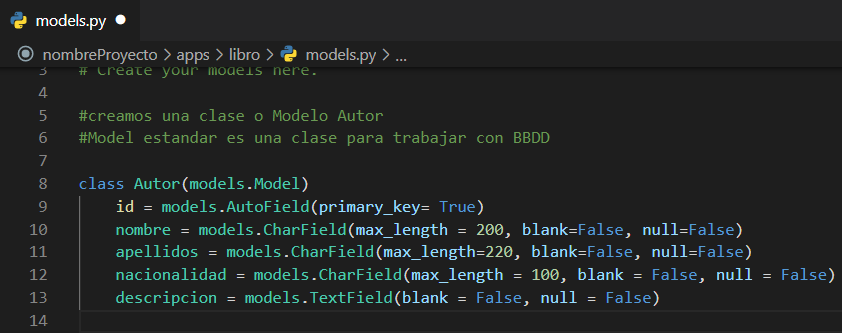
Un usuario interactua con una pagina Web o Sitio creado con Django la cual sera capturada por una URL. Esta ultima estará conectada con una VISTA. Esta ultima a diferencia del MVC tendrá el código (la lógica de programacion) que va a recibir la interacción que el usuario ha hecho y en base a ese pedido, le avisa al MODELO (quien esta conectado con la BBDD). El Modelo le responde a la VISTA. Esta ultima lo que recibió del modelo lo va a enviar a un TEMPLATE, el cual es el que le llega al usuario en definitiva.

**INTRODUCCION A MODELOS** (En las clases siguientes se profundizara, aquí solo mostramos como crear una clase Modelo)

**Modelos:** Se van a basar en clases las cuales van a ser la representación de las tablas de BBDD.

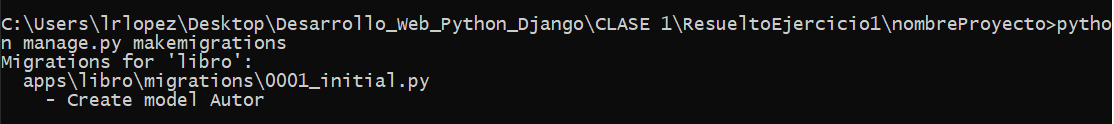
**Pasos para crear una Biblioteca**

1. Creamos nuestro modelo. Clase Autor

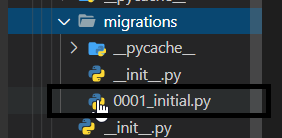


2. Hacemos que nuestro modelo se cree o impacte en la BBDD

python manage.py makemigrations (desde el cmd, misma ruta que manage.py)

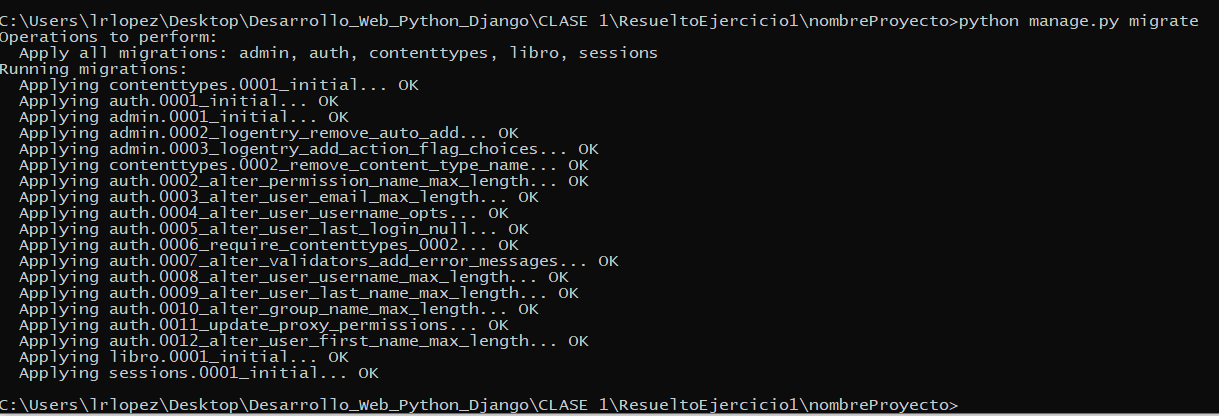


Se creo el archivo identificando los modelos que existen en todo el proyecto y dejando una estructura que toma Django para que posteriormente sea convertida en código SQL para aplicarse en BBDD.



3. Se aplican los modelos en BBDD

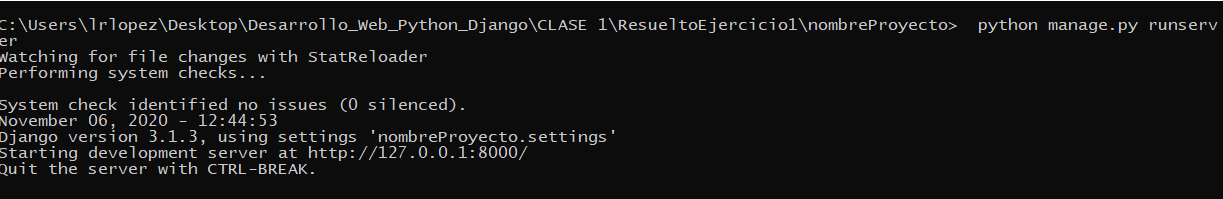
python manage.py migrate



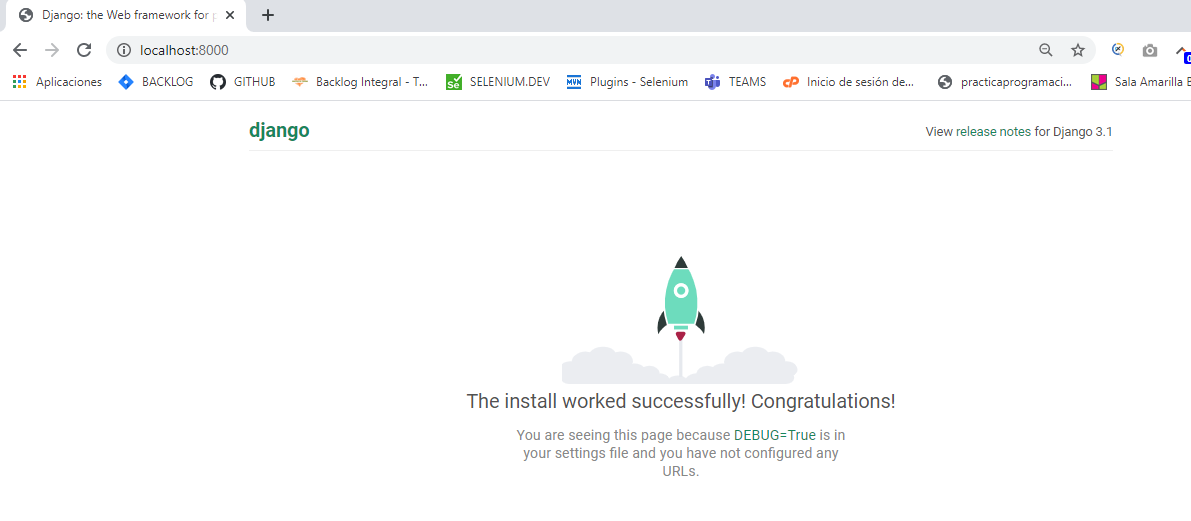
Con esto ya tenemos los cambios aplicados.

4. Comprobamos desde Django si se aplico en BBDD (sin necesidad de entrar a la base). Esta herramienta es el administrador de Django.

a. Ejecutamos el servidor de Django. => python manage.py runserver



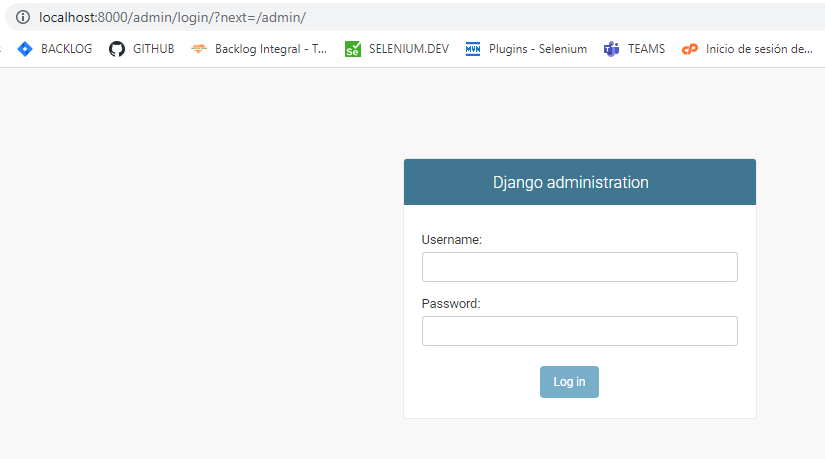
b. Nos vamos al navegador y colocamos <http://localhost:8000/>



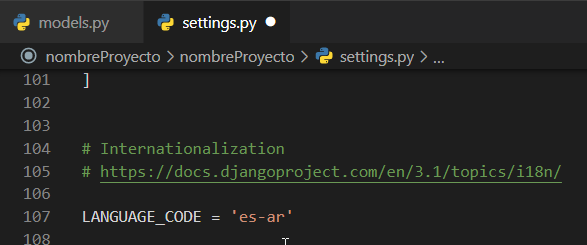
Pagina por default que se levanta en django.

**CONSOLA ADMINISTRATIVA DE DJANGO.**

c. Administrador de Django: Vamos a <http://localhost:8000/admin/>



Para cambiar el idioma vamos a:



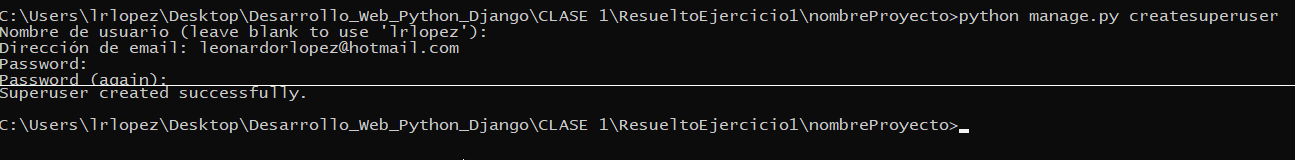
Es:español – ar: Argentina



d. Creamos un usuario para el admin.

python manage.py createsuperuser

contraseña: tar.nuf.roi-2



e. python manage.py runserver

f. Ingresamos al administrador con las credenciales creadas.



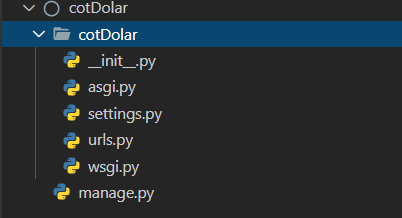
El administrador de Django ofrece una administración para el super usuario en donde podemos hacer de todo. Crear usuarios, administrarlos, etc.

**TRAER LA COTIZACION DEL DOLAR**

**Ejercicio 1 LABORATORIO:**

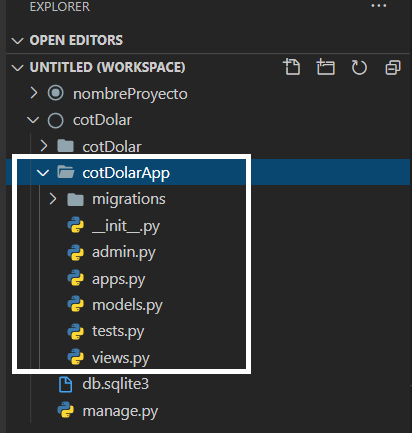
**a. Creamos nuestro Proyecto.** Nos ubicamos sobre una carpeta y ejecutamos:

django-admin startproject cotDolar

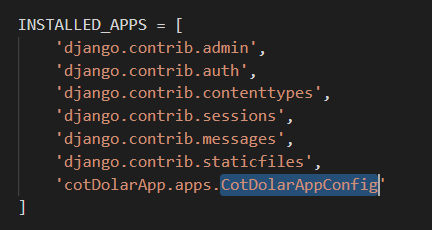
****

b. Creamos nuestra app Cotizacion del Dólar:

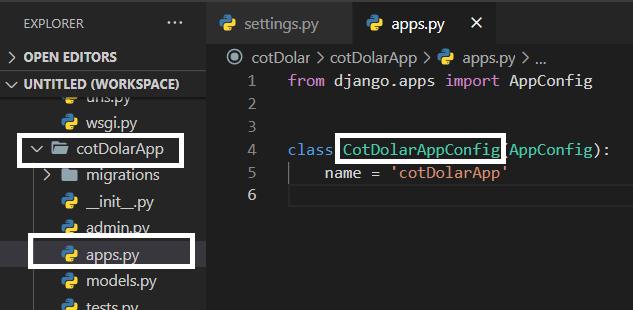
python manage.py startapp cotDolarApp



d. Agregamos la aplicación cotDolarApp al proyecto cotDolar, editamos settings.py



En donde la línea agregada esta relacionada con la clase del archivo cotDolarApp/apps.py:



Notas en relación a esta configuración:

Para **Django**  un **Project** es una aplicación web de Django. Por otra parte el termino **aplicación**  describe un paquete de **Python**  que provee ciertas características relacionadas a 1 aplicación. Esta ultima puede ser reutilizada en varios projects.

Las aplicaciones se linkean a un Project gracias al seteo del **INSTALLED\_APPS** y también por otros mecanismos.

Cada aplicación mantiene metadata en la clase **AppConfig** que se encuentra en el archivo **apps.py**. Por medio de la cual cada proyecto puede acceder a los componentes de cada aplicación.

El atributo **name** que se incluye dentro del **AppConfig** determina que aplicación interviene en la configuración del AppConfig.

**INSTALLED\_APP**: Representa una lista de strings que designan aplicaciones. Cada línea es un path por puntos o (dotted Python path) hacia la aplicación correspondiente.

e. Creamos una función dentro de nuestro archivo cotDolarApp/views.py:

Borramos todo lo anterior y escribimos:

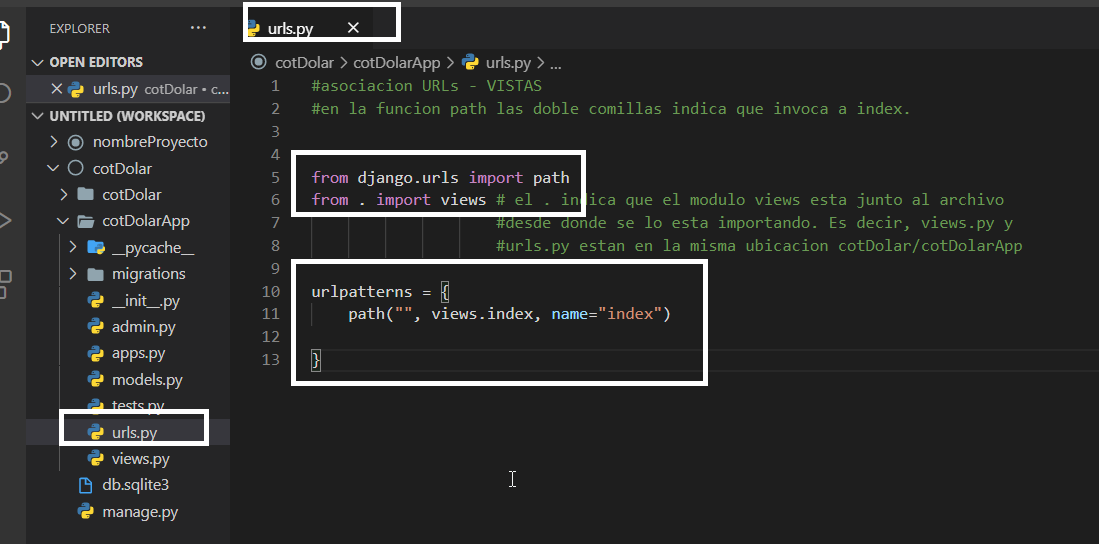
import requests

from django.http import HttpResponseRedirect

def index(request):

    return HttpResponseRedirect("https://api.recursospython.com/dollar")

f. Le indicamos a Django a que direccion de URL debe responder la función index(). Para eso creamos el archivo cotDolarApp/urls.py con el siguiente código:



g. Seteo a nivel Proyecto – archivo urls.py

Agregamos la configuración URL – VISTAS particular de la aplicación cotDolarApp a la configuración general del proyecto cotDolar modificando el archivo urls.py del proyecto cotDolar de la siguiente manera:



Con esto le decimos a Django que todas las dirección de URL que comienzan con cotDolarApp/ (<http://127.0.0.1:8000/cotDolarApp/> o <http://dominio.com/cotDolarApp>) sean derivadas a la aplicación cotDolarApp. Lo que siga a esa dirección de URL (por ejemplo /cotizacionVenta) será administrado por la configuración de las direcciones URL de la aplicación, es decir por cotDolarApp/urls.py. Alli indicamos que nuestra vista index() debia responder con el sitio <https://api.recursospython.com/dollar> . Al visitar el sitio <http://127.0.0.1:8000/cotDolarApp/> veremos el contenido de la respuesta que retornamos en esa función index() (vista).

Previamente migramos

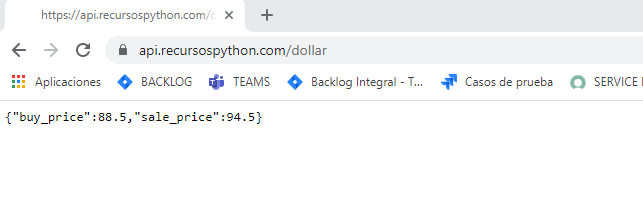
python manage.py migrate

Iniciamos el servidor

python manage.py runserver

Y vamos a la siguiente pagina

<http://127.0.0.1:8000/cotDolarApp/>



h. Agregamos una nueva configuracion en urls.py y una nueva función en views.py:

from django.urls import path

from . import views

urlpatterns = {

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

    path("cotizacion-dolar", views.cotizacion\_dolar, name="cotizacion\_dolar")

}

**cotización\_dolar**

import requests

from django.http import HttpResponseRedirect, HttpResponse

def index(request):

    return HttpResponseRedirect("https://api.recursospython.com/dollar")

def cotizacion\_dolar(request):

    r = requests.get("https://api.recursospython.com/dollar")

    resultado = r.json()

    html = f"""

        <html>

        <title>Cotizacion del dolar</title>

        <p><strong>Compra</strong>: {resultado["buy\_price"]}</p>

        <p><strong>Venta</strong>: {resultado["sale\_price"]}</p>

        </html>

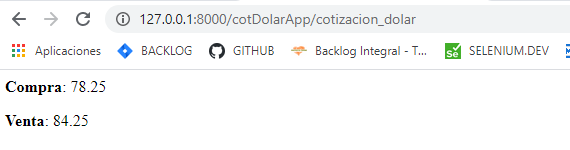
        """

    return HttpResponse(html)

Reiniciamos nuevamente el servidor y cumplimos con lo que pide el ejercicio 1:

C:\Users\lrlopez\Desktop\Desarrollo\_Web\_Python\_Django\CLASE 1\PracticaEnClase\cotDolar> **python manage.py runserver**

<http://127.0.0.1:8000/cotDolarApp/cotizacion-dolar>





**EJERCICIO 2 LABORATORIO**

**Parte 1:**

**a. Creamos nuestro Proyecto.** Nos ubicamos sobre una carpeta y ejecutamos:

django-admin startproject ejercicio2

b. Iniciamos el sitio:

cd cotDolar

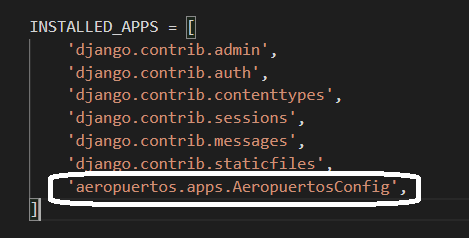
python manage.py runserver

<http://127.0.0.1:8000/> => para ver que este arriba nuestro proyecto

c. Creamos nuestra app aeropuertos:

python manage.py startapp aeropuertos

d. Agregamos la aplicación aeropuertos al proyecto ejercicio2, editamos settings.py



En donde la línea agregada esta relacionada con la clase del archivo aeropuertos/apps.py:

python manage.py runserver (iniciamos nuevamente el servidor)

e. Creamos una función dentro de nuestro archivo aeropuertos/views.py:

Borramos todo lo anterior y escribimos:

from django.http import HttpResponse

#Primera parte del ejercicio 2

def aeropuertos(request):

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

    html = """

        <html>

        <title>Lista de aeropuertos</title>

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

            <thead>

                <tr>

                    <th>Aeropuerto</th>

                    <th>Ciudad</th>

                    <th>Pais</th>

                </tr>

            </thead>

    """

    for linea in f:

        datos = linea.split(",")  #separa el texto de acuerdo al parametro de split, si esta vacio significa que toma los espacios

        nombre = datos[1].replace('"', "") #reemplaza las comillas dobles por un espacio, en la 2da columna (despues de dividida)

        ciudad = datos[2].replace('"', "") #idem pero en la 3er columna

        pais = datos[3].replace('"', "") #idem pero en la 4ta columna

        html += f"""

            <tr>

                <td>{nombre}</td>    <!--Agrega el nombre en la 1er celda-->

                <td>{ciudad}</td>    <!--Agrega la ciudad en la 1er celda-->

                <td>{pais}</td>      <!--Agrega el pais en la 1er celda-->

            </tr>

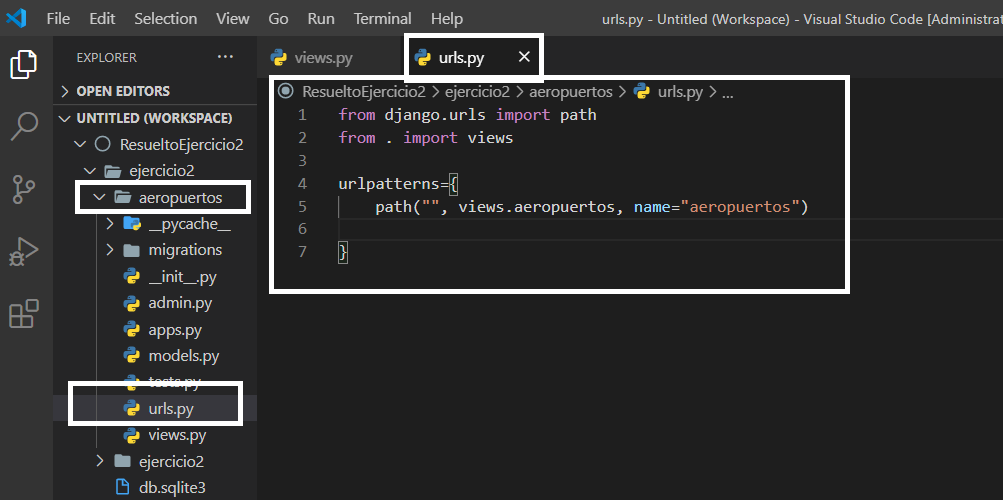
        """

    f.close()    #cierra el archivo

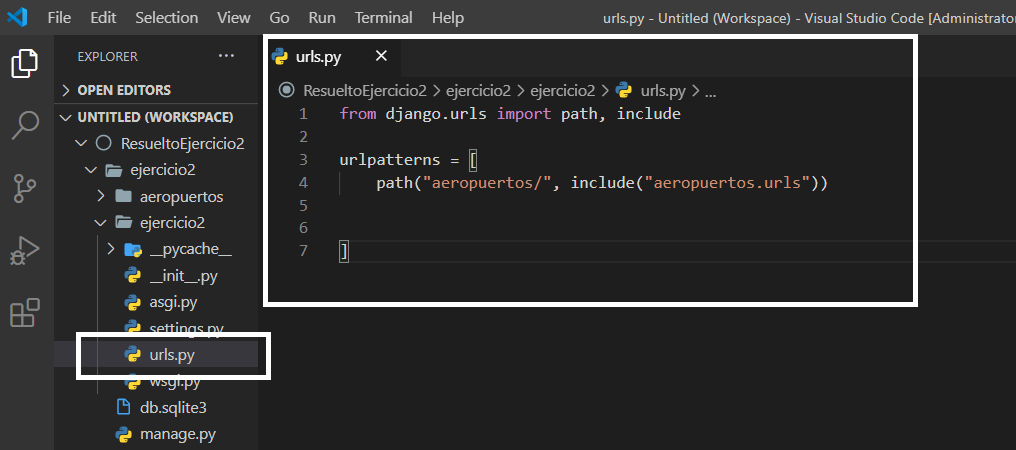
    html += "</table></html>" #cierra la tabla a nivel html

    return HttpResponse(html) #devuelve el html creado previamente

f. Le indicamos a Django a que direccion de URL debe responder la función aeropuertos (). Para eso creamos el archivo aeropuertos/urls.py con el siguiente código:



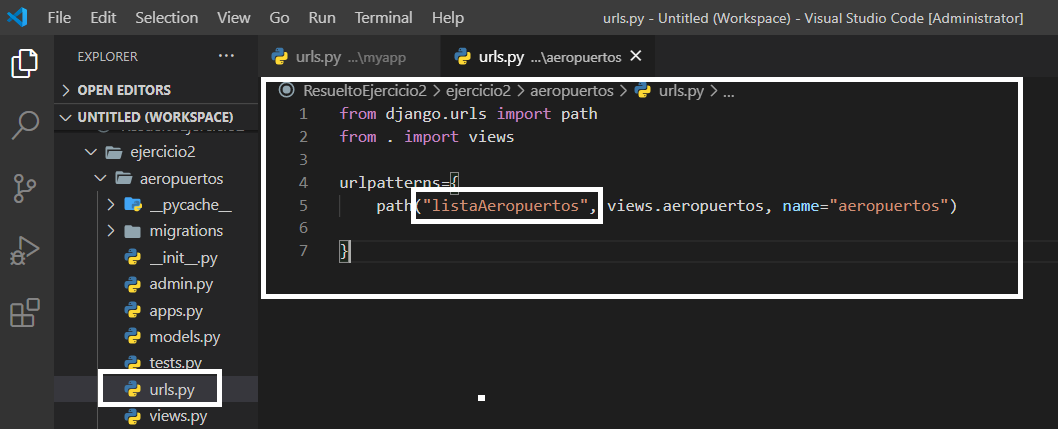
g. Agregamos la configuración URL – VISTAS particular de la aplicación aeropuertos a la configuración general del proyecto ejercicio2 modificando el archivo urls.py del proyecto ejercicio2 de la siguiente manera:



Con esto le decimos a Django que todas las dirección de URL que comienzan con aeropuertos/ (<http://127.0.0.1:8000/aeropuertos/> o <http://dominio.com/aeropuertos>) sean derivadas a la aplicación aeropuertos. Lo que siga a esa dirección de URL (por ejemplo /preciosPasajes) será administrado por la configuración de las direcciones URL de la aplicación, es decir por aeropuertos/urls.py. Alli indicamos que nuestra vista o funcion aeropuertos() debia responder con el Excel en formato html. Al visitar el sitio <http://127.0.0.1:8000/aeropuertos/> veremos el contenido de la respuesta que retornamos.

**Nota:** Recordar ubicar el archivo .csv en la misma ruta donde se encuentra manage.py

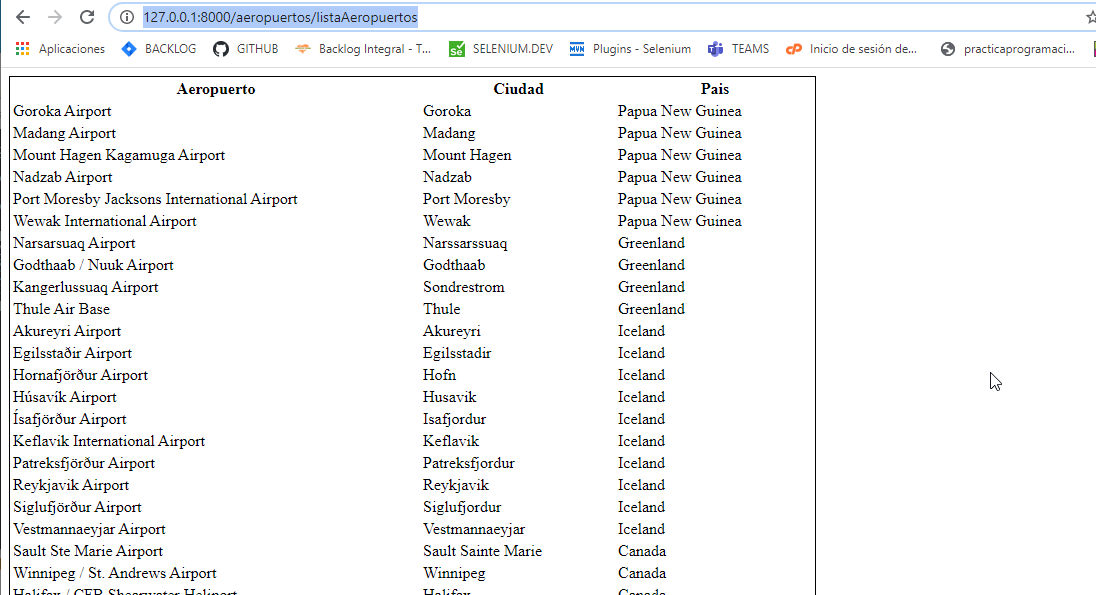
h. hacemos unas pequeñas modificaciones en urls.py:



Reiniciamos nuevamente el servidor y cumplimos con lo que pide el ejercicio 2 parte 1:

python manage.py runserver

http://127.0.0.1:8000/aeropuertos/listaAeropuertos



**Parte 2: Devolver la misma información pero en formato JSON**

**a. Creamos nuestro Proyecto. >** Ya esta creado

b. Iniciamos el sitio: >Ya esta iniciado

c. Creamos nuestra app aeropuertos: > Ya esta creada

d. Agregamos la aplicación aeropuertos al proyecto ejercicio2, editamos settings.py > Ya esta hecho

e. Creamos una función dentro de nuestro archivo aeropuertosJson/views.py:

(recordamos importar JasonResponse package)

#Segunda parte del ejercicio 2

def aeropuertosJson(request):

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

    aeropuertos= [] #definimos una lista vacia

    for linea in f:

        datos = linea.split(",")

        aeropuerto= {               #creamos un diccionario

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

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

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

        }

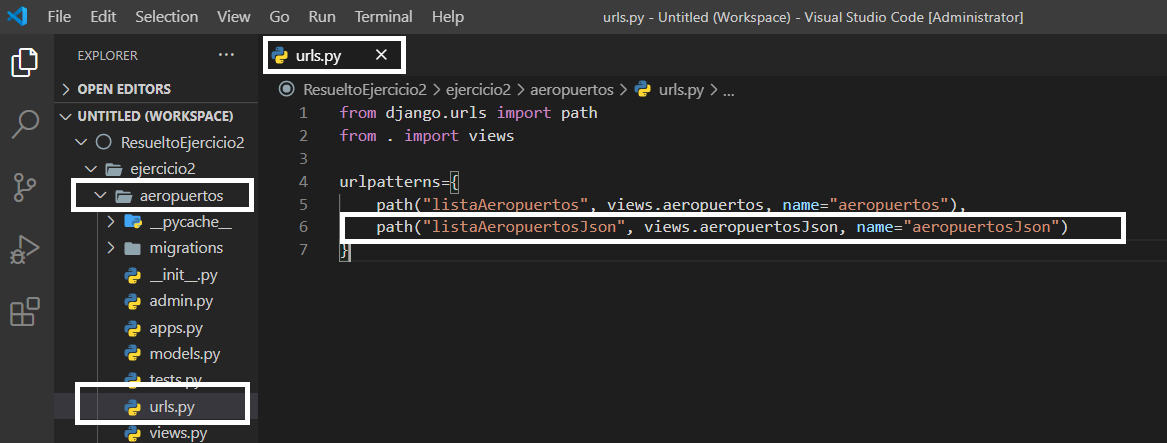
        aeropuertos.append(aeropuerto) #agregamos el diccionario a la lista

    f.close()

    return JsonResponse(aeropuertos, safe=False) #safe=False significa que cualquier objeto puede ser serializado para ser respondido

                                                 #safe=True solo el diccionario puede ser serializado

f. Le indicamos a Django a que direccion de URL debe responder la función aeropuertosJson (). Para eso en el archivo aeropuertos/urls.py agregamos el siguiente codigo:



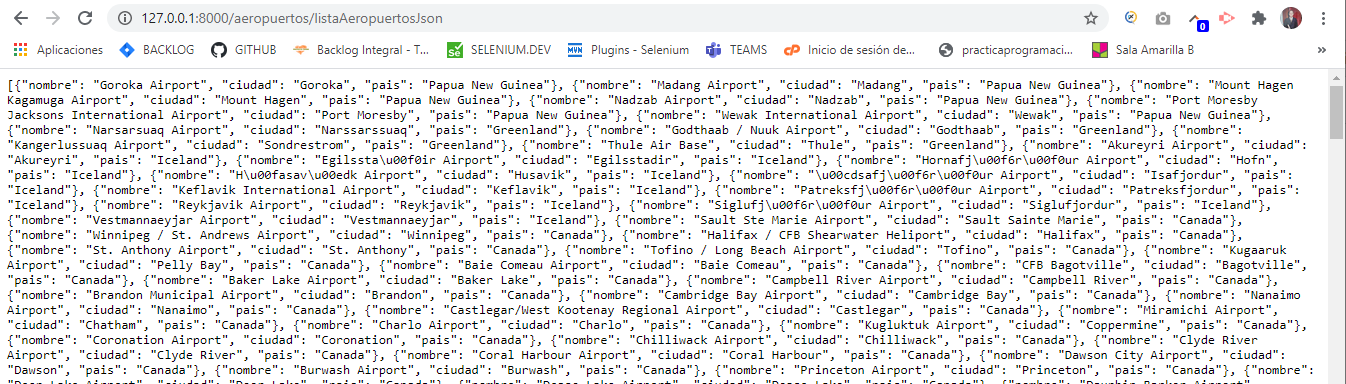
g. Agregamos la configuración URL – VISTAS particular de la aplicación aeropuertos a la configuración general del proyecto ejercicio2 modificando el archivo urls.py del proyecto ejercicio2 > **Ya esta hecho**

h. hacemos unas pequeñas modificaciones en urls.py (del proyecto): **>Ya esta hecho**

Reiniciamos nuevamente el servidor y cumplimos con lo que pide el ejercicio 2 parte 2:

C:\Users\lrlopez\Desktop\Desarrollo\_Web\_Python\_Django\CLASE 1\ResueltoEjercicio2\ejercicio2> python manage.py runserver

http://127.0.0.1:8000/aeropuertos/listaAeropuertosJson



Codigo:



Prueba desde la consola interactiva de Python

1. Ubicarse en la ruta del proyecto e iniciarlo.

C:\Users\lrlopez\Desktop\Desarrollo\_Web\_Python\_Django\CLASE 1\ResueltoEjercicio2\ejercicio2>python manage.py runserver

1. Realizar los siguientes pasos:

C:\Users\lrlopez>python

Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>> import requests

>>> r = requests.get("http://127.0.0.1:8000/aeropuertos/listaAeropuertosJson")

>>> aeropuertos = r.json()

>>> aeropuertos[0]

{'nombre': 'Goroka Airport', 'ciudad': 'Goroka', 'pais': 'Papua New Guinea'}

>>>

**Ejercicio Integrador:**

**a. Creamos nuestro Proyecto.** Nos ubicamos sobre una carpeta y ejecutamos:

django-admin startproject indumentariOnline

b. Iniciamos el sitio:

cd indumentariaonline

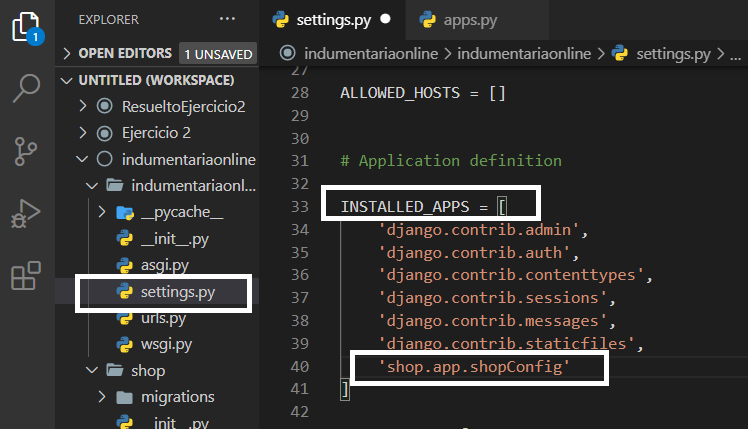
python manage.py runserver

<http://127.0.0.1:8000/> => para ver que este arriba nuestro proyecto

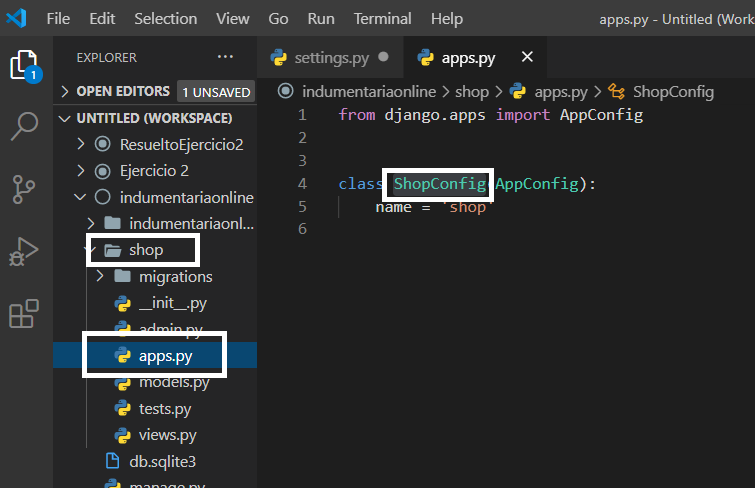
c. Creamos nuestra app shop:

python manage.py startapp shop

d. Agregamos la aplicación shop al proyecto indumentariaonline, editamos settings.py del proyecto:



En donde la línea agregada esta relacionada con la clase del archivo shop/apps.py:



python manage.py runserver (iniciamos nuevamente el servidor)

e. Creamos una función (Vista) dentro de nuestro archivo shop/views.py:

1. index()

2. contacto()

Borramos todo lo anterior y escribimos:

from django.http import HttpResponse

def index(request):

    html = """

        <html>

            <head>

                <title>Indumentaria Online </title>

            </head>

            <body>

                <h1>¡Bienvenido a nuestro sitio</h1>

                <p> <a href="">Ir a contacto</a> </p>

            </body>

        </html>

    """

    return HttpResponse(html)

def contacto(request):

    html = """

        <html>

            <title>Contacto - Indumentaria Online </title>

            <h1> Contacto </h1>

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

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

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

            <p>

                <a href="/shop"> Volver al inicio </a>

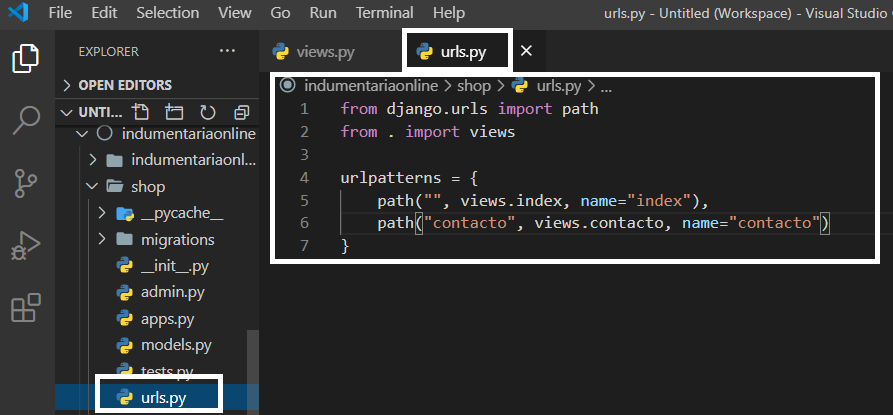
            <p>

        </html>

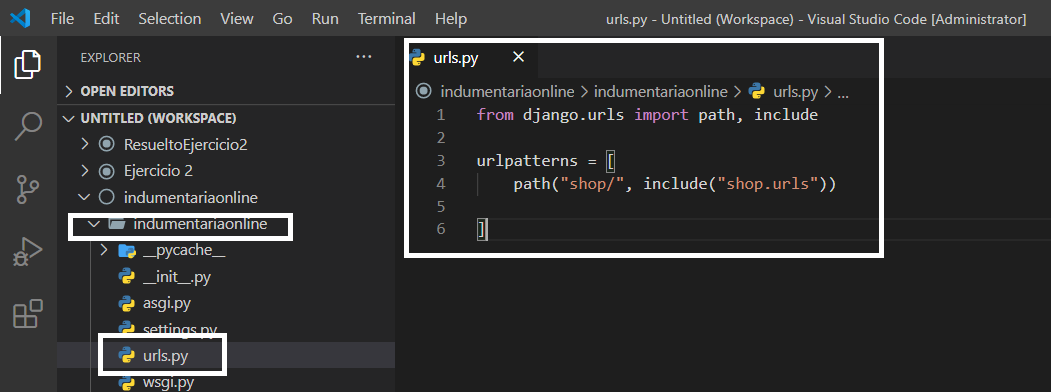
    """

    return HttpResponse(html)

f. Le indicamos a Django a que direccion de URL debe responder la función index(). Para eso creamos el archivo shop/urls.py con el siguiente código:



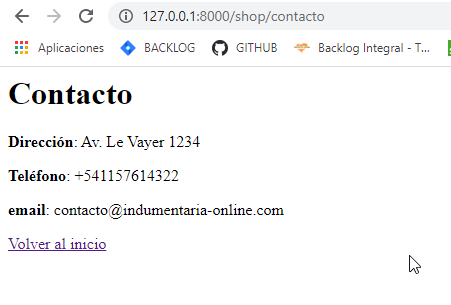
g. Agregamos la configuración URL – VISTAS particular de la aplicación shop a la configuración general del proyecto indumentariaonline modificando el archivo urls.py del proyecto de la siguiente manera:



Con esto le decimos a Django que todas las dirección de URL que comienzan con shop/ (<http://127.0.0.1:8000/shop/> o <http://dominio.com/shop>) sean derivadas a la aplicación shop. Lo que siga a esa dirección de URL (por ejemplo /Compras) será administrado por la configuración de las direcciones URL de la aplicación, es decir por shop/urls.py. Alli indicamos que nuestra vista index() debia responder con el sitio



Y si damos click en el link “Ir a contacto” debemos obtener la siguiente salida:



h. Codigo Resuelto:

