Informe de Laboratorio 07

Tema: Python - Django

Nota

Estudiante	Escuela	Asignatura
Jose Alonzo Gordillo Mendoza jgordillome@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Web 2 Semestre: III Código: 20220577

Laboratorio	Tema	Duración
07	Python - Django	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - A	Del 29 de Junio 2023	Al 14 Julio 2023

1. Tarea

- Recreacion de los videos proporcionados en el aula virtual sobre relaciones one to many y many to many, asi como la generacion de pdf's y el envio de emails.

2. URL de Repositorio Github

- URL para el laboratorio 07 en el Repositorio GitHub.
- `https://github.com/JoseGordilloMendoza/LAB07.git`

3. Ejercicios

3.1. Estructura de laboratorio 07

- La distribucion de archivos sera la siguiente (teniendo en cuenta los archivos mas importantes, por ejemplo las aplicaciones o los cambios que signifiquen acciones importantes):

```
lab07/
    |----informeLatex/
    |----lab07relations/
        |----modelExa
            |----__pycache__
            |----migrations
            |----__init__.py
            |----admin.py
            |----apps.py
            |----models.py
            |----tests.py
            |----views.py
        |----lab07relations
            |----__pycache__
            |----migrations
            |----__init__.py
            |----asgi.py
            |----settings.py
            |----manage.py
            |----urls.py
            |----wsgi.py
        |----db.sqlite3
        |----manage.py

    |----lab07emails_pdf/
        |----emails
            |----emails
                |----__pycache__
                |----__init__.py
                |----asgi.py
                |----settings.py
                |----urls.py
                |----wsgi.py
            |----send
                |----__pycache__
                |----migrations
                |----templates
                |----__init__.py
                |----admin.py
                |----models.py
                |----apps.py
                |----tests.py
                |----urls.py
                |----views.py
            |----db.sqlite3
            |----manage.py
        |----pdfs
            |----pdfs
                |----__pycache__
                |----__init__.py
                |----asgi.py
                |----settings.py
                |----urls.py
                |----wsgi.py
            |----pdf
                |----__pycache__
```

```
            |----migrations
            |----templates
            |----__init__.py
            |----admin.py
            |----apps.py
            |----models.py
            |----tests.py
            |----utils.py
            |----views.py
        |----db.sqlite3
        |----manage.py
```

3.2. Relaciones One to many - Many to Many

En primer lugar veamos lo mas resaltante de modelsExa/apps.py:

Listing 1: Analizando modelsExa/apps.py

```python
from django.apps import AppConfig

class ModelsexaConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'modelsExa'
```

Este archivo de configuración de la aplicación ModelsexaConfig establece la configuración específica para la aplicación llamada 'modelsExa', incluyendo la configuración del campo de clave primaria automática.

Listing 2: models.py del proyecto

```python
from django.db import models

    class Simple(models.Model):
        text = models.CharField(max_length=10)
        number = models.IntegerField(null=True)
        url = models.URLField(default="www.example.com")

        def __str__(self):
            return self.url
    class DateExample(models.Model):
        the_date = models.DateTimeField

    class NullExample(models.Model):
        col = models.CharField(max_length=10, blank=True, null=True)

    class Language(models.Model):
        name = models.CharField(max_length=10)

        def __str__(self):
            return self.name

    class Framework(models.Model):
        name = models.CharField(max_length=10)
        language = models.ForeignKey(Language, on_delete=models.CASCADE)
```

```
        def __str__(self):
            return self.name

    class Movie(models.Model):
        name = models.CharField(max_length=10)

        def __str__(self):
            return self.name

    class Character(models.Model):
        name = models.CharField(max_length=10)
        movies = models.ManyToManyField(Movie)

        def __str__(self):
         return self.name
```

Estas clases de modelos definen las estructuras y relaciones de base de datos para diferentes entidades, como "Simple", "DateExample", "NullExample", "Language", "Framework", "Moviez Çharacter". Cada clase de modelo se mapea a una tabla en la base de datos y los campos de los modelos se traducen en columnas en esas tablas. Las relaciones entre los modelos se establecen mediante campos como ForeignKey y ManyToManyField.

- lab07_one_to_many /modelsExample

Listing 3: asgi.py

```
import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'modelsExample.settings')

application = get_asgi_application()
```

Este código configura el entorno y obtiene una instancia del servidor de aplicaciones ASGI para la aplicación Django definida en el módulo de configuración 'modelsExample.settings'. Esta instancia del servidor de aplicaciones se utiliza para manejar las solicitudes y respuestas HTTP de la aplicación cuando se ejecuta utilizando un servidor compatible con ASGI.

Listing 4: settings.py

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

Al incluir 'modelsExa' en INSTALLED_APPS, me aseguro de que Django reconozca y utilice la aplicación personalizada en el proyecto. Esto permite que los modelos y otros componentes de la aplicación estén disponibles y se integren con el resto del sistema de Django.

Listing 5: wsgi.py

```
import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'modelsExample.settings')

application = get_wsgi_application()
```

Este código configura el entorno y obtiene una instancia del servidor de aplicaciones WSGI para la aplicación Django definida en el módulo de configuración 'modelsExample.settings'.

3.3. Generaciónn de pdf

- Analizando la configuracion del proyecto

Listing 6: Archivo settings.py

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

En este fragmento vemos lo mas resaltante de settings.py, pues tenemos una aplicacion llamada pdf de la cual obtendremos la principal logica de nuestra pagina.

- Analizando las url del proyecto

Listing 7: Archivo urls.py

```
from django.contrib import admin
from django.urls import path
from pdf.views import GeneratePdf

urlpatterns = [
    path('admin/', admin.site.urls),
    path('pdf/',GeneratePdf.as_view(), name='PDF'),
]
```

Como vemos tendremos una url "pdf"que sera la que imprima un pdf como tal haciendo uso de una view; esto teniendo en cuenta que importaremos las urls de la aplicacion pdf.

- En la app pdf, analizaremos el archivo utils.py

Listing 8: Archivo pdf/utils.py

```
from io import BytesIO
from django.http import HttpResponse
```

```
from django.template.loader import get_template

from xhtml2pdf import pisa

def render_to_pdf(template_src, context_dict={}):
    template = get_template(template_src)
    html  = template.render(context_dict)
    result = BytesIO()
    pdf = pisa.pisaDocument(BytesIO(html.encode("ISO-8859-1")), result)
    if not pdf.err:
        return HttpResponse(result.getvalue(), content_type='application/pdf')
    return None
```

En esta función toma una plantilla HTML, la renderiza con los datos del contexto y la convierte en un archivo PDF utilizando xhtml2pdf. Luego, devuelve una respuesta HTTP que contiene el PDF generado. Hay que saber que utils.py es un archivo comúnmente utilizado en proyectos Django para almacenar funciones y clases de utilidad que se utilizan en varias partes del proyecto. Es una práctica común separar estas funciones y clases en un archivo aparte para mantener un código más organizado y modular.

- Vistas de la app

Listing 9: Archivo pdf/views.py

```
from django.shortcuts import render

from django.http import HttpResponse
from django.template.loader import get_template
from django.views.generic import View

from .utils import render_to_pdf #created in step 4

class GeneratePdf(View):
    def get(self, request, *args, **kwargs):
        template = get_template('invoice.html')
        context = {
             'today': "Today",
             'amount': 1339.99,
            'customer_name': 'Cooper Mann',
            'invoice_id': 1233434,
        }
        html = template.render(context)
        pdf = render_to_pdf('invoice.html',context)
        if pdf:
            response = HttpResponse(pdf, content_type='application/pdf')
            filename = "Invoice_%s.pdf"%("12341231")
            content = "inline; filename='%s'"%(filename)
            download = request.GET.get("download")
            if download:
                content = "attachment; filename='%s'" %(filename)
            response['Content-Disposition']=content
            return response

        return HttpResponse("Not Found")
```

Esta vista basada en clase permite generar y devolver un archivo PDF a partir de una plantilla HTML cuando se realiza una solicitud HTTP GET a la vista. El archivo PDF puede ser descargado o mostrado en el navegador según los parámetros proporcionados en la URL de la solicitud. A resaltar que se importa la función render_to_pdf desde un archivo utils.py, que se supone que contiene una función personalizada para generar un archivo PDF a partir de una plantilla HTML.

- HTML de la app

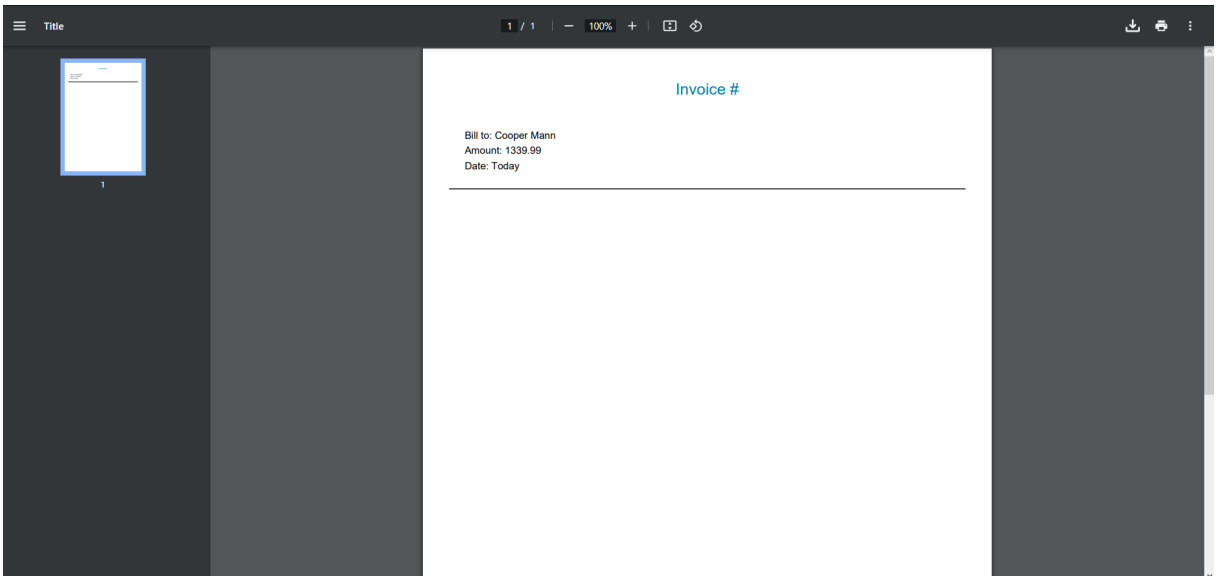
Listing 10: Archivo pdf/templates/invoice.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>Title</title>
        <style type="text/css">
            body {
                font-weight: 200;
                font-size: 14px;
            }
            .header {
                font-size: 20px;
                font-weight: 100;
                text-align: center;
                color: #007cae;
            }
            .title {
                font-size: 22px;
                font-weight: 100;
               /* text-align: right;*/
               padding: 10px 20px 0px 20px;
            }
            .title span {
                color: #007cae;
            }
            .details {
                padding: 10px 20px 0px 20px;
                text-align: left !important;
                /*margin-left: 40%;*/
            }
            .hrItem {
                border: none;
                height: 1px;
                /* Set the hr color */
                color: #333; /* old IE */
                background-color: #aca0a0; /* Modern Browsers */
            }
        </style>
    </head>
    <body>
        <div class='wrapper'>
            <div class='header'>
                <p class='title'>Invoice # </p>
            </div>
        <div>
```

```
            <div class='details'>
                Bill to: {{customer_name}}<br/>
                Amount: {{amount}}<br/>
                Date: {{today}}
                <hr class='hrItem' />
            </div>
        </div>
        </body>
    </html>
```

Esta plantilla HTML define la estructura y los estilos básicos para generar un documento con un encabezado, detalles y una línea horizontal para separar secciones. Los datos del cliente, el monto y la fecha se insertarán en los lugares correspondientes utilizando variables del contexto cuando se renderice la plantilla.

- Demostracion en navegador



En la captura vemos el pdf que se genera al entrar a la url .../pdf, cabe resaltar que la direccion predeterminada que se suele usar .../ no esta definida, por lo que el pdf se generera en la de pdf.

3.4. Envio de email

- Analizando la configuracion del proyecto

Listing 11: Archivo settings.py

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

```
]

...

EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587

EMAIL_HOST_USER ='sjbalonzo201c@gmail.com'
EMAIL_HOST_PASSWORD = 'ifhkdehiucgqrcmg'
EMAIL_USE_TLS = True
EMAIL_USE_SSL = False
```

En este fragmento vemos lo mas resaltante de settings.py, pues tenemos una aplicacion llamada send de la cual obtendremos la principal logica de nuestra pagina. Por otra parte configuramos lo concerniente al correo, estas configuraciones se utilizan en Django para enviar correos electrónicos a través del servidor SMTP de Gmail. Asegúrate de proporcionar correctamente tu dirección de correo electrónico y contraseña en EMAIL_HOST_USER y EMAIL_HOST_PASSWORD, respectivamente, para permitir que Django se autentique en el servidor SMTP de Gmail y envíe correos electrónicos desde esa cuenta. Es muy importante mencionar que el password se configura en la cuenta de gmail habilitando/generando una çontraseña de aplicaciones"pues con la contraseña usual nos saltara un error.

- Analizando las url del proyecto

Listing 12: Archivo urls.py

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

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

Como vemos tendremos una url que sera la que envie un email como tal haciendo uso de una view; esto teniendo en cuenta que importaremos las urls de la aplicacion send, que es la que contiene la url original.

- En la app send, analizaremos el archivo urls.py

Listing 13: Archivo send/urls.py

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

urlpatterns = [

    path('', views.index),
]
```

Esta configuración de URLs indica que cuando se acceda a la ruta principal del sitio, se ejecutará la vista index definida en el módulo views. Puedes agregar más objetos path a la lista urlpatterns para mapear otras rutas de URL a diferentes vistas en tu aplicación Django.

- Analizaremos el archivo views.py

Listing 14: Archivo send/views.py

```
from django.shortcuts import render
from django.core.mail import send_mail

# Create your views here.
def index(request):
    send_mail('EMAIL DESDE DJANGO',
              'Probando envio de email desde DJANGO, hola :D',
              'sjbalonzo201c@gmail.com',
              ['jgordillome@unsa.edu.pe'],
              fail_silently=False)
    return render(request, 'send/index.html')
```

Esta vista en Django envía un correo electrónico utilizando la función send_mail y luego renderiza una plantilla HTML llamada 'send/index.html'. Podemos personalizar los detalles del correo electrónico, como el asunto, el cuerpo y las direcciones de los remitentes y destinatarios, según las necesidades.

- Analizaremos el archivo templates/send/index.html

Listing 15: Archivo send/templates/send/index.html

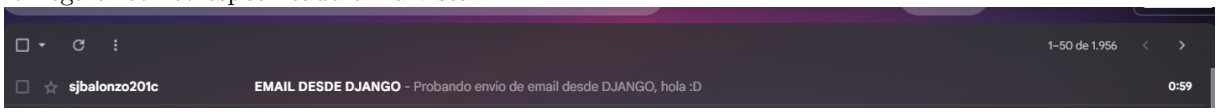
```
<!DOCTYPE html>
<html lang="en">

  <head>
    <title>Sent!</title>
  </head>
  <body>
    <h1>sent an email!</h1>
  </body>

</html>
```

Eesta plantilla HTML muestra un mensaje simple "sent an email!.[en] el cuerpo del documento. Usada para mostrar una confirmación de que se ha enviado correctamente un correo electrónico desde una aplicación Django, por ejemplo, después de llamar a la función send_mail en una vista y redirigir a esta plantilla.

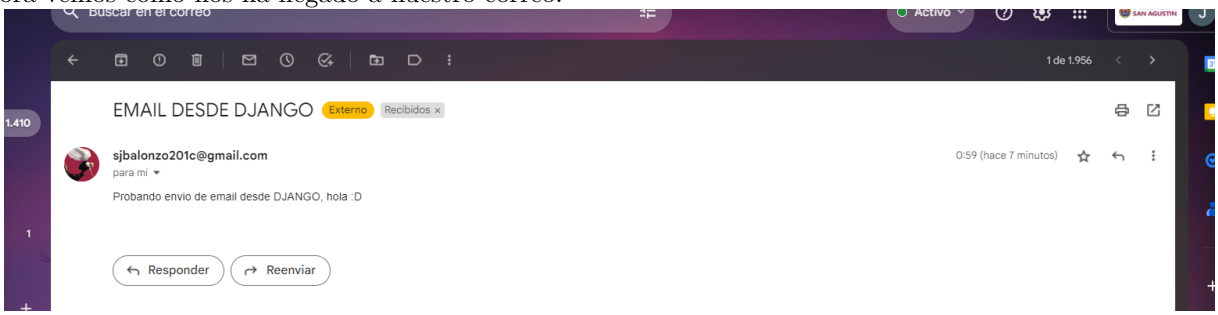
- Demostracion en navegador

sent an email!

En la captura vemos que la pagina mostrandonos un mensaje de que se envio el email; por tanto veamos como llega al correo especificado en la vista.



Ahora vemos como nos ha llegado a nuestro correo.



Como vemos nos muestra lo que especificamos anteriormente en nuestra view, es decir: asunto, mensaje, remitente y destinario.

Link donde esta alojado el video (FlipGrid):
GRUPO: `https://flip.com/8a44775d`
VIDEO ESPECIFICO: `https://flip.com/groups/14644257/topics/37115679/responses/430300075/comments`

4. Cuestionario

- Sin cuestionario -

5. Conclusiones

- Django proporciona un sólido sistema de relaciones que permite establecer vínculos entre diferentes modelos. Las relaciones .°ne-to-many"permiten establecer una conexión en la que un modelo está relacionado con varios modelos de otro tipo. Las relaciones "many-to-many"permiten una mayor flexibilidad, ya que un modelo puede estar vinculado a múltiples instancias de otro modelo y viceversa.
- En las relaciones "many-to-many", Django utiliza una tabla intermedia para almacenar los vínculos entre los modelos. Esta tabla contiene claves foráneas que apuntan a las instancias de los modelos relacionados. Por ejemplo, si tenemos un modelo de .Estudiantez un modelo de Çurso", la tabla intermedia contendría claves foráneas que enlazarían estudiantes con cursos y viceversa.
- Django ofrece una variedad de bibliotecas y herramientas que hacen posible transformar contenido HTML o datos estructurados en archivos PDF con el fin de crear archivos de archivo PDF. ReportLab y WeasyPrint son dos bibliotecas destacadas que se utilizan en Django. Estas bibliotecas brindan herramientas para crear documentos PDF personalizados, incluida la capacidad de incluir estilos, imágenes y tablas.
- Cuando se trata de enviar correos electrónicos, Django tiene una API de correo electrónico integrada que agiliza el proceso. Proporciona configuraciones para el servidor de correo, como SMTP o configuraciones de correo específicas del proveedor, y proporciona una interfaz simple para crear y enviar correo electrónico. Además, es posible adjuntar archivos PDF u otros tipos de archivos a los mensajes de correo electrónico enviados mediante Django.
- Finalmente Django proporciona herramientas y capacidades para crear rápida y fácilmente archivos PDF y enviar correo electrónico. Esto permite a los desarrolladores incorporar funciones como la generación de facturas en PDF y la entrega de correo electrónico, entre otras posibilidades, en sus aplicaciones en línea desarrolladas por Django.

6. Referencias

- https://www.w3schools.com/python/python_reference.asp
- https://docs.djangoproject.com/en/4.2/topics/email/
- https://www.scaler.com/topics/django/relationships-in-django-models/
- https://docs.djangoproject.com/en/4.2/howto/outputting-pdf/