Informe de Laboratorio 07

Tema: Relaciones de uno a muchos, muchos a muchos e impresión de pdf y emails

Nota

Estudiante	Escuela	Asignatura
William Isaias Roque Quispe wroquequi@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Web 2 Semestre: III

Laboratorio	Tema	Duración
07	Relaciones de uno a muchos, muchos a muchos e impresión de pdf y emails	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - A	Del 29 junio 2023	Al 14 de julio 2023

1. Tarea

- Reproducir las actividades de los videos donde trabajamos:
  1. Relación de uno a muchos
  2. Relación muchos a muchos
  3. Impresión de pdfs
  4. Envío de emails

2. Equipos, materiales y temas utilizados

- Sistema Operativo
- GNU Vim.
- Python 3.11.3.
- Git
- Cuenta en GitHub con el correo institucional.
- Entorno virtual.
- Django 4.
- xhtml2pdf 0.2.11.

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/WilliamIsaiasRoque/Lab07-Pweb2.git`
- URL del video subido en Flip.
- `https://flip.com/groups/14644257/topics/37115679/responses`

4. Actividades

4.1. Crear un directorio para el laboratorio

- En este directorio inicializaremos el git y agregaremos un primer commit para probar.

Listing 1: Creando directorio de trabajo y accediendo en él

```
$ mkdir PWEB2lab/Lab07-Pweb2/
```

Listing 2: Dirijiéndonos al directorio de trabajo

```
$ cd PWEB2lab/Lab07-Pweb2/
```

Listing 3: Inicializando el repositorio Git y agregando remote add origin

```
$ git init
$ git remote add origin https://github.com/WilliamIsaiasRoque/Lab07-Pweb2.git
```

Listing 4: Añadiendo un README.md como primer push

```
$ echo "# Lab07-Pweb2" >> README.md
$ git add README.md
$ git commit -m "Initial commit"
$ git push -u origin main
```

4.2. Crear un entorno virtual en el directorio

Listing 5: Creando el directorio django_env y accediendo en esa carpeta

```
$ virtualenv env
```

Listing 6: Activando el entorno virtual con source

```
$ source env/Scripts/activate
```

Listing 7: Instalando Django y xhtml2pdf (es una herramienta que convierte documentos XHTML a formato PDF.). Luego guardamos las versiones en un requirements.txt

```
$ pip install Django
$ pip install xhtml2pdf
```

```
$ pip list
Package   Version
---------- -------
asgiref   3.7.2
Django    4.2.2
sqlparse  0.4.4
tzdata    2023.3
xhtml2pdf  0.2.11
$ pip freeze > requirements.txt
```

Listing 8: Creando tres proyectos Django para las tres actividades.

```
$ django-admin startproject model_examples
$ django-admin startproject pdfs_examples
$ django-admin startproject emailexample
```

4.3. Solución de ejercicios y commits

- Los commits más importantes aparecerán en esta sección, al final se mostrarán unas capturas con todos los commits.
- **PROYECTO model_examples**
- Creamos una app con 'python manage.py startapp example'.
- Accedemos a los settings.py del proyecto y agregamos 'example' a INSTALLED APPS.
- Se agregaron distintos modelos al archivo 'models.py' los cuales hacen referencia a una tabla en la base de datos, se procede a explicar cada uno:
- El modelo Simple posee tres campos (text, number y url), con el método __str__ devuelve la URl almacenada.
- El modelo DateExample tiene un campo llamado 'the_date', que almacena una fecha y hora usando DateTimeField().
- El modelo NullExample tiene un área llamada col, encargada de almacenar texto con una longitud máxima de 10 caracteres.
- Para la relación uno a muchos se crearon los modelos Language y Framework, Language es capaz de guardar hasta 10 caracteres y Framework establece una relación de clave externa (ForeignKey) con el modelo Language.
- Para la relación muchos a muchos se adicionaron los modelos Movie y Character, Movie almacena texto hasta un límite de 10 caracteres y Character tiene un campo 'movies' que establece la relación 'ManyToManyField' con el modelo anterior.

Listing 9: model_examples/example/models.py

```
1 from django.db import models
2
3 class Simple(models.Model):
4     text = models.CharField(max_length=10)
5     number = models.IntegerField(null=True)
6     url = models.URLField(default='www.example.com')
```

```
7
8       def __str__(self):
9           return self.url
10
11  class DateExample(models.Model):
12      the_date = models.DateTimeField()
13
14  class NullExample(models.Model):
15      col = models.CharField(max_length=10, blank=True, null=True)
16
17  class Language(models.Model):
18      name = models.CharField(max_length=10)
19
20      def __str__(self):
21          return self.name
22
23  class Framework(models.Model):
24      name = models.CharField(max_length=10)
25      language = models.ForeignKey(Language, on_delete=models.CASCADE)
26
27      def __str__(self):
28          return self.name
29
30  class Movie(models.Model):
31      name = models.CharField(max_length=10)
32
33      def __str__(self):
34          return self.name
35
36  class Character(models.Model):
37      name = models.CharField(max_length=10)
38      movies = models.ManyToManyField(Movie)
39
40      def __str__(self):
41          return self.name
```

- Se ejecuta 'python manage.py makemigrations' para crear archivos de migración y después 'python manage.py migrate' para aplicar los cambios a la base de datos.
- En admin.py se registran todos los modelos creados anteriormente para que puedan ser utilizados correctamente.
- El resto de archivos no han sufrido cambios durante la elaboración de los modelos, por ello no se han agregado al informe.

Listing 10: model_examples/example/admin.py

```
1  from django.contrib import admin
2  from .models import Simple, DateExample, NullExample, Character, Movie
3
4  admin.site.register(Simple)
5  admin.site.register(DateExample)
6  admin.site.register(NullExample)
7  admin.site.register(Character)
8  admin.site.register(Movie)
```

- Se crearon diversas tablas usando los modelos a través de 'python manage.py shell'
- Para probar la relación uno a muchos, se creó la tabla example_language con los elementos Python y Java, la tabla example_framework posee los elementos Django, Flask, Bootle y Spring.

Table: example_language

	id	name
	Filter	Filter
1	1	Python
2	2	Java

Tabla example_character

Table: example_framework

	id	name	language_id
	Filter	Filter	Filter
1	1	Django	1
2	2	Flask	1
3	3	Bootle	1
4	4	Spring	2

Tabla example_character

- Se realizaron consultas (query) para recoletar los datos de las tablas. Para ello se volvió a acceder a 'Python manage.py shell'.

Listing 11: Query One To Many

```
from example.models import Language, Framework
Framework.objects.all()
Framework.objects.filter(language__name='Python')
Framework.objects.filter(language__name='Java')
Framework.objects.filter(language__name__startswith='Py')
Framework.objects.filter(language__name__startswith='Ja')
Language.objects.filter(framework__name='Spring')
Language.objects.filter(framework__name='Django')
```

- Para probar la relación muchos a muchos, se creó la tabla example_character con los elementos Captain America y Thor, la tabla example_movie contiene un listado de películas (Avengers, Civil War, Thor: Dark World y Winter Soldier).
- La tabla example_character_movies demuestra que un actor puede estar asociado a varias películas y viceversa.

Table: example_character

	id	name
	Filter	Filter
1	1	Captain America
2	2	Thor

Tabla example_character

Table: example_movie

	id	name
	Filter	Filter
1	1	Avengers
2	2	Civil War
3	3	Thor: Dark World
4	4	Winter Soldier

Tabla example_movie

Table: example_character_movies

	id	character_id	movie_id
	Filter	Filter	Filter
1	1	1	1
2	2	1	2
3	3	2	1
4	4	2	3
5	5	1	4

Tabla example_character_movies

- Se realizaron otras consultas (query) para recoletar los datos de las tablas mediante shell.

Listing 12: Many To Many Query

```
from example.models import Character, Movie
Character.objects.filter(movies__name='Civil War')
Character.objects.filter(movies__name='Avengers')
Movie.objects.filter(character__name='Captain America')
Movie.objects.filter(character__name='Thor')
captain_america = Character.objects.get(name='Captain America')
captain_america
captain_america.movies.all()
avengers = Movie.objects.get(name='Avengers')
avengers
avengers.character_set.all()
```

- **PROYECTO pdfs_examples**
- Para este proyecto no será necesario crear una aplicación extra ya que se agregarán los archivos requeridos directamente en la carpeta base.
- Se agrega 'utils.py' para convertir un archivo HTML en un archivo PDF utilizando la biblioteca xhtml2pdf. Si no hay errores durante la conversión del HTML a PDF, se devuelve un objeto HttpResponse que contiene los datos del PDF generado, con el tipo de contenido establecido como 'application/pdf'.

Listing 13: pdfs_examples/utils.py

```python
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
```

- Después se ha modificado la ruta de TEMPLATES DIRS en settings.py para que utilice la carpeta templates que se creará pronto.
- Se debe agregar el archivo views.py el cual se encargará de generar un archivo PDF, cuando se accede a la vista a través de una solicitud GET, se carga la plantilla HTML, se renderiza con los datos del contexto y termina generándose el PDF. Si se ha creado correctamente, devuelve una respuesta HTTP adjunta según los parámetros de la solicitud GET. En caso contrario, retorna una respuesta indicando que no se encontró el elemento.
- Para urls.py se debe agregar 'pdf/' para generar y devolver un archivo PDF utilizando la vista GeneratePdf.

Listing 14: pdfs_examples/views.py

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

from .utils import render_to_pdf

class GeneratePdf(View):
    def get(self, request, *args, **kwargs):
        template = get_template('pdf/invoice.html')
        context = {
            'invoice_id': 123456,
            'customer_name': 'Ricardo Milos',
            'amount': 1981.99,
            'today': 'Today',
```

```
        }
        html = template.render(context)
        pdf = render_to_pdf('pdf/invoice.html', context)
        if pdf:
            response = HttpResponse(pdf, content_type='application/pdf')
            filename = "Invoice_%s.pdf" %("666")
            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")
```

Listing 15: pdfs_examples/urls.py

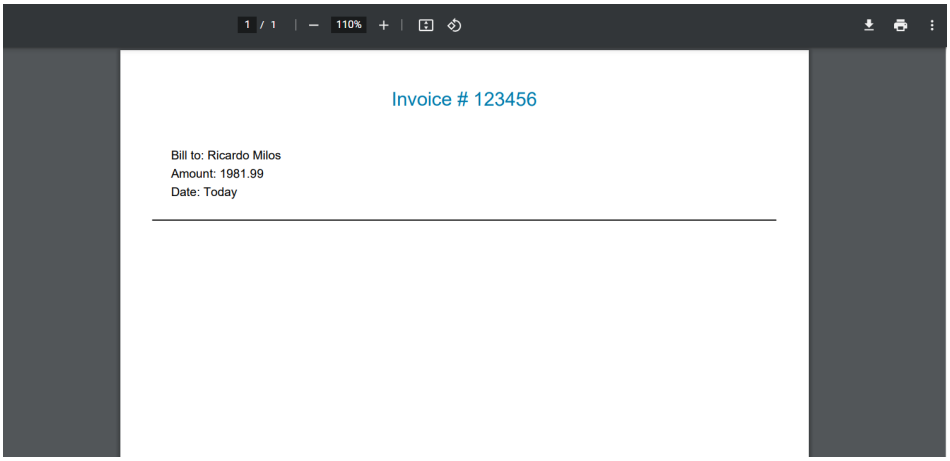
```
"""
URL configuration for pdfs_examples project.

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
    1. Add an import: from my_app import views
    2. Add a URL to urlpatterns: path('', views.home, name='home')
Class-based views
    1. Add an import: from other_app.views import Home
    2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path
from .views import GeneratePdf

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

- Es importante crear la carpeta templates, acceder y crear otra carpeta llamada pdf, dentro de esta nueva carpeta se inicializará un index.html, el cual tiene un estilo y estructura predeterminada para mostrar información de una factura. Los valores de invoice_id, customer_name, amount y today se insertarán en los lugares correspondientes cuando se renderice la plantilla con datos específicos.

Listing 16: pdfs_examples/templates/pdf/invoice.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>Impresin de pdf</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: #fff; /* Modern Browsers */
            }
        </style>
    </head>
    <body>
        <div class='wrapper'>
            <div class='header'>
                <p class='title'>Invoice # {{ invoice_id }}</p>
            </div>
        <div>
        <div class='details'>
            Bill to: {{ customer_name }}<br/>
            Amount: {{ amount }}<br/>
            Date: {{ today }}
            <hr class='hrItem' />
        </div>
    </div>
    </body>
</html>
```



Ejecución del servidor

- **PROYECTO emailexample**
- Primero se inicia una app con 'python manage.py startapp send', después se accede a settings.py para agregar a 'send' en INSTALLED APPS.
- Luego se agregó el archivo urls.py dentro de la carpeta send.

Listing 17: emailexample/send/urls.py

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

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

- Se procede a modificar el urls.py del proyecto (es distinto al de send).

Listing 18: emailexample/emailexample/urls.py

```python
"""
URL configuration for emailexample project.

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
    1. Add an import: from my_app import views
    2. Add a URL to urlpatterns: path('', views.home, name='home')
Class-based views
    1. Add an import: from other_app.views import Home
    2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
```

```
    2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include

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

- Dentro de la carpeta send se crea un directorio llamado templates, se accede a él y se crea otro directorio llamado send al cual debemos acceder nuevamente, para finalmente crear un index.html el cual tiene de título 'Sent!' y muestra en pantalla un mensaje confirmando el envío del email.

Listing 19: emailexample/send/templates/send/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" contents="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" contents="ie=edge">
    <title>Sent!</title>
</head>
<body>
    <h1>Sent an email!</h1>
</body>
</html>
```

- En settings.py se agrega la información necesaria para enviar correos electrónicos utilizando el servidor SMTP de Gmail. Se especifica el servidor, el puerto, la dirección de correo electrónico y la contraseña del remitente. También se establece si se utilizará TLS para la conexión segura.

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = 'static/'


EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_HOST_USER = 'willroqlab@gmail.com'
EMAIL_HOST_PASSWORD = 'bfwx ebfz suge axai'
EMAIL_USE_TLS = True
EMAIL_USE_SSL = False

```

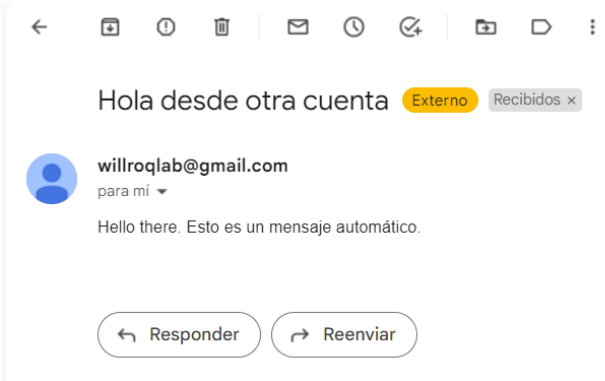
Configuración de emails

- En views.py se define una vista que envía un correo electrónico utilizando la función 'send_mail' y luego muestra una página HTML mediante la función render. El correo electrónico tiene un asunto, un cuerpo y se envía desde la dirección de correo electrónico especificada al destinatario indicado. Después de enviar el correo electrónico, se muestra una página HTML al usuario.

Listing 20: emailexample/send/views.py

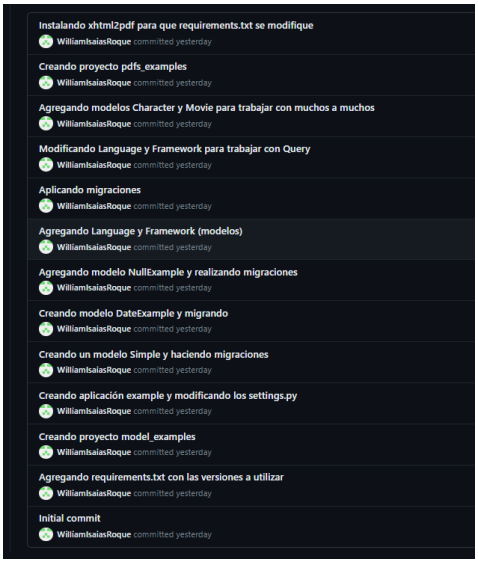
```python
from django.shortcuts import render
from django.core.mail import send_mail

def index(request):
    send_mail('Hola desde otra cuenta',
    'Hello there. Esto es un mensaje automtico.',
    'willroqlab@gmail.com',
    ['wroquequi@unsa.edu.pe'],
    fail_silently=False)

    return render(request, 'send/index.html')
```

Ejecución del servidor

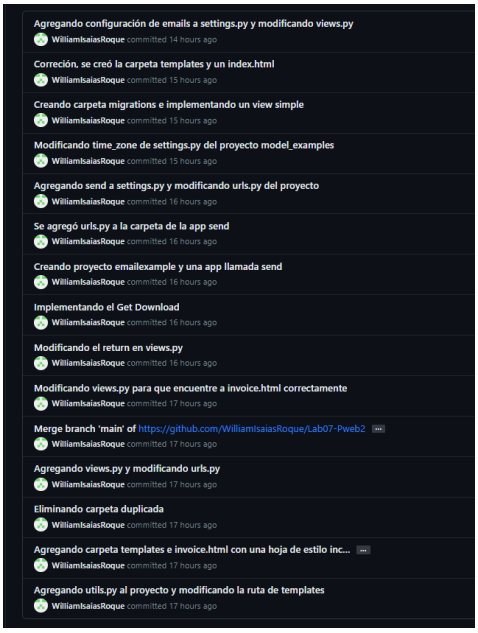


Mensaje recibido en la bandeja de entrada

- A continuación una captura de todos los commits realizados en el repositorio GitHub:



Primera parte de los commits



Segunda parte de los commits

5. Conclusiones

- Django ofrece un enfoque sencillo para gestionar relaciones One-to-Many entre modelos. Con la utilización de campos ForeignKey o OneToManyField, se facilita la creación y manipulación de relaciones entre tablas.
- Django se encarga automáticamente de la creación de tablas intermedias para almacenar los vínculos entre los objetos relacionados de muchos a muchos con ManyToManyField.
- Con el uso de bibliotecas como ReportLab, xhtml2pdf o WeasyPrint, se pueden generar archivos PDF dinámicamente a partir de plantillas o datos y servirlos como una respuesta HTTP.
- A través del módulo django.core.mail, es posible configurar los parámetros de un servidor SMTP, establecer los destinatarios, asunto, cuerpo y adjuntos de los correos electrónicos y enviar emails.

6. Referencias

- `https://www.pythontutorial.net/django-tutorial/django-one-to-many/`
- `https://docs.djangoproject.com/en/4.2/topics/db/examples/many_to_many/`
- `https://www.codingforentrepreneurs.com/blog/html-template-to-pdf-in-django/`
- `https://docs.djangoproject.com/en/4.2/topics/email/`