# GUÍA DE LABORATORIO 07

## Tema: Uno a muchos, pdf y emails con django

| Nota |
|---|
| |

| Estudiante | Escuela | Asignatura |
|---|---|---|
| Carrasco Choque Arles Melvin | Escuela Profesional de Ingeniería de Sistemas | Programacion Web<br>Semestre: III<br>Código: |

| Laboratorio | Tema | Duración |
|---|---|---|
| 07 | Uno a muchos, pdf y emails con django | 02 horas |

| Semestre académico | Fecha de inicio | Fecha de entrega |
|---|---|---|
| 2023 - A | 8 Julio 2023 | 13 Julio 2023 |

# 1. TAREA:

- 1. Relación de uno a muchos
- 2. Relación muchos a muchos
- 3. Impresión de pdfs
- 4. Envio de emails
- Crear su video Flipgrid:

# 2. URL DEL REPOSITORIO EN GITHUB Y DEL VIDEO FLIPGRID:

- Repositorio GITHUB: `https://github.com/carrascoArles/pwb-07`
- video Flipgrid: `https://flip.com/s/me1zmyzMbzhY`

# 3. DESARROLLO DE LA ACTIVIDAD:

## 3.1. Uno a muchos y muchos a muchos

Para el uno a muchos se crearon dos clases, Language y Frameworks, en modo de padre e hijos es como lo vamos a usar.

Se observa que Language guarda solo su atributo nombre, pero la clase Framework usa uno mas siendo laguage, lo que va a hacer es concatenarse a modo de pertenencia a Language con el ForeignKey.

Listing 1: models.py de modelsExa)

```
    from django.db import models


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
```

Para el "muchos a muchos", se crearon dos clases Movie y Character, movie guarda solo su nombre y en este caso el modificado seria el Character.

Character usa el atributo movies, que a diferencia que el de uno a muchos, usa el ManytoManyField, concatenando a Movie, haciendo un llamado multiple entre los dos.

Listing 2: models.py de modelsExa)

```
   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
```

## 3.2. PDF

En este codigo es gonde se genera el pdf, el template busca la plantilla con el templatesrc, una vez renderizado nos va a dar un html, finalmente el BytesIO va a servir para almacenar el resultado en formato PDF.

El modulo "pisa.[es] el mas importante ya que este lo convierte en pdf, recibiendo el html y renderizando a un pdf en este result.

Listing 3: 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
```

Ahora con el views nos va a mostrar el pdf, en su respectivo apartado, con el template obtenemos el invoice y le ponemos el context que se recolecta en el html, luego se manda en pdf con el render to pdf (definido en utils.py).

Si es que no hay problemas que llama el response, y se descargar con el download.

Listing 4: 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")
```

## 3.3. EMAIL

Lo mas importante es importar send-mail ya que de ahi se va a sacar todo, primero en la funcion de send-mail, se usan 5 parametros, el titulo,el contenido, el gmail personal, el mail al que vamos enviar y el fail-silently que su funcion es avisar si algo falla y no se envio.

Retorna un index.html en el cual solo va a llamar si no hubo errores.

Listing 5: views.py de send)

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

# Create your views here.
def index(request):
    send_mail('Hello from prettyPrinted',
```

```
            'hello there. this us an automated message',
            'tilina.com',
            ['gokas38987@kameili.com'],
            fail_silently=False)
    return render(request, 'send/index.html')
```

Para finalizar se hicieron ajustes importantes en el setting del proyecto para que pueda enviar. Tenemos el host que es a la empresa que pertenece nuestro correo, el port que depende tambien del host, el user ,contraseña, el tls y el ssl que tambien depende de cada host.

Listing 6: settings.py de emailexample)

```
  EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_HOST_USER ='esatilina.com'
EMAIL_HOST_PASSWORD = 'aea'
EMAIL_USE_TLS = True
EMAIL_USE_SSL = False
```