Informe de Laboratorio 08

Tema: Django

Nota

Estudiante	Escuela	Asignatura
Victor Gonzalo Maldonado Vilca vmaldonadov@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Programación Web 2 Semestre: III Código: 1702122

Tarea	Tema	Duración
08	Django	2 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2024 - A	Del 9 de abril de 2024	Al 15 de junio de 2024

1. Introducción

Django es un poderoso framework de desarrollo web en Python que simplifica la gestión de bases de datos a través de su sistema ORM. Este sistema incluye la capacidad de manejar relaciones de uno a muchos y muchos a muchos, además de generar PDFs y enviar correos electrónicos. Las relaciones de uno a muchos se implementan utilizando ForeignKey, lo que permite que una fila de una tabla esté vinculada a múltiples filas de otra tabla. Por otro lado, las relaciones de muchos a muchos se manejan con ManyToManyField, lo que permite interacciones complejas entre tablas. Para la generación de PDFs, se pueden usar bibliotecas como WeasyPrint que transforman plantillas HTML en documentos PDF. El envío de correos electrónicos es facilitado por django.core.mail, permitiendo enviar correos electrónicos de manera programada. Estas funcionalidades hacen de Django una herramienta versátil y eficiente para el desarrollo de aplicaciones web completas.

2. Objetivos

- Implementar en una aplicación en Django el manejo de Bases de datos.
- Utilizar una tabla y relacionarla con muchas tablas.
- Utilizar muchas tablas y relacionarlas con muchas tablas.
- Implementar el envío de emails y la impresión de pdfs desde una aplicación Django

3. Tarea

- Actividades
  1. Crear un proyecto en Django
  2. Siga los pasos de los videos para poder implementar la aplicación de Relación de Uno a muchos en una Base de Datos, muchos a muchos, impresión de pdfs y envío de emails.
  3. Use git y haga los commits necesarios para manejar correctamente la aplicación.
- Ejercicios Propuestos
  1. Deberán replicar la actividad de los videos donde se trabaja con Relacion de uno a muchos, de muchos a muchos, impresión de pdfs y envío de emails; adecuándolo desde un proyecto en blanco Django.
  2. Para ello crear una carpeta dentro del proyecto github colaborativo con el docente, e informar el link donde se encuentra.
  3. Eres libre de agregar CSS para decorar tu trabajo.
  4. Ya sabes que el trabajo con Git es obligatorio. Revisa los videos entregados.

4. Entregables

- Informe hecho en Latex
- URL: Repositorio GitHub
- Link de Vídeo Explicativo (Youtube o flipgrid)

5. Equipos, materiales y temas utilizados

- Proyectos de Django
- Aplicaciones en Django
- Relaciones entre Tablas
- Envío de Emails e impresión de PDFs

6. URL de Repositorio Github

- URL del Repositorio GitHub
- `https://github.com/Victor-Gonzalo-Maldonado-Vilca/Django_Lab08.git`

7. Link de Video

- Link del Vídeo Explicativo
- **Youtube:** `https://www.youtube.com/watch?v=0j3u3PcELEk`
- **flipgrid:** `https://flip.com/s/YSxoPgTJ6DyZ`

8. Metodología

8.1. Creación del entorno de Trabajo

8.1.1. Configuración de la carpeta de trabajo

Listing 1: Creación del Directorio

```
mkdir lab08 && cd lab08
virtualenv -p python3 .
Scripts/activate
pip install Django
```

8.1.2. Creación carpeta del proyecto

Listing 2: Carpeta src

```
mkdir src && cd src
```

8.1.3. Inicializar git

Listing 3: Inicializar git

```
git init
```

8.1.4. Crear .gitignore

Seguimos el siguiente Repositorio `https://github.com/django/django/blob/main/.gitignore`

8.2. Creacion del proyecto y apps

8.2.1. Crear Proyectos

Listing 4: Crear proyecto

```
django-admin startproject model_examples
django-admin startproject pdf_printing
django-admin startproject email_project
```

8.2.2. Crear App en el Proyecto model examples

Listing 5: Crear App

```
  python manage.py startapp example
```

8.2.3. Crear App en el Proyecto email project

Listing 6: Crear App

```
python manage.py startapp email_app
```

8.3. Uso de Xampp como base de Datos

Se especificará como se logro en desarrollo de trabajo del video 5 'Database-settings'

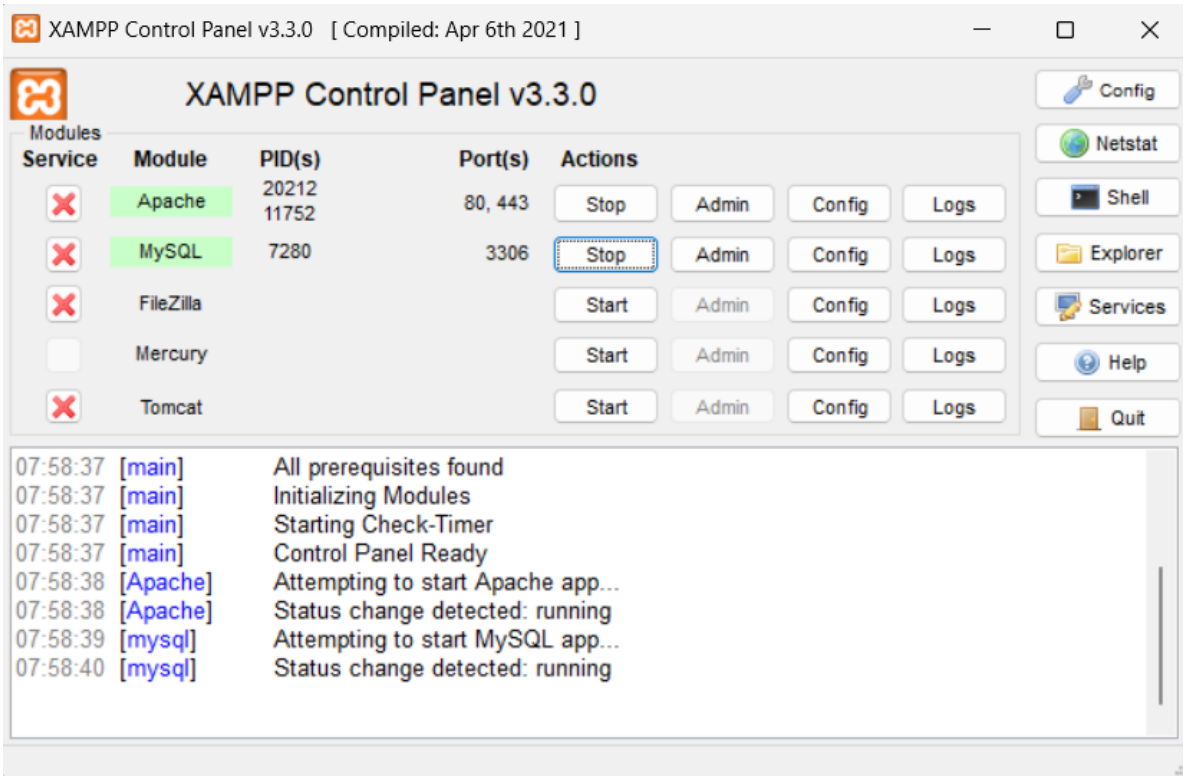

Figura 1: Xampp Panel Control

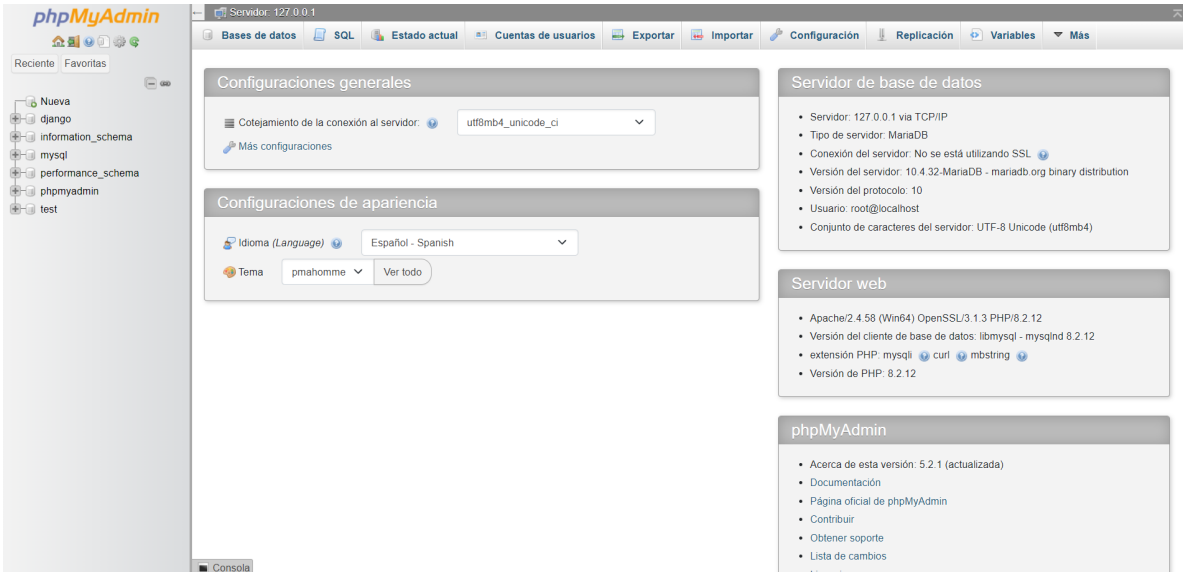

Figura 2: mySQL - Xampp

9. Desarrollo del trabajo

9.1. Relación de uno a muchos:

9.1.1. Vídeo 1: one-to-many-relationships (Proyecto 'model examples')

- **Modelos – Descripcion:** Se define dos modelos en Django que son Language y Framework. El modelo Language tiene un solo campo name que es un CharField con un máximo de 10 caracteres. La representación en forma de cadena del objeto Language devuelve el valor del campo name. El modelo Framework también tiene un campo name que es un CharField con un máximo de 10 caracteres y un campo language que es una clave foránea (ForeignKey) referenciando el modelo Language. La opción on delete=models.CASCADE significa que cuando un objeto Language es eliminado, también se eliminarán todos los objetos Framework asociados a ese lenguaje. La representación en forma de cadena del objeto Framework devuelve el valor del campo name.

```
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
```

- **Shell – Descripción:** Lo expresado en el video ilustra cómo manejar relaciones de uno a muchos en Django creando y vinculando objetos de dos modelos distintos: Language y Framework. Inicialmente, se crea una instancia de Language llamada 'Python' y se guarda en la base de datos. Luego, se crean varias instancias de Framework (Django, Flask y Bottle) y se asocian con

la instancia de 'Python'. Cada Framework se guarda en la base de datos después de establecer la relación. Más tarde, se crea otra instancia de Language llamada 'Java' y se guarda. Finalmente, se crea una instancia de Framework llamada 'Spring', se asocia con 'Java' y se guarda en la base de datos. Este proceso muestra cómo establecer y guardar relaciones de uno a muchos en Django, donde varios frameworks están asociados con un solo lenguaje.

```
>>>from example.models import Language, Framework
>>>python = Language(name='Python')
>>>python.save()
>>>django = Framework(name='Django')
>>>flask = Framework(name='Flask')
>>>python
<Language: Language object (1)>
>>>django.language = python
>>>flask.language = python
>>>bottle = Framework(name='Bottle', language=python)
>>>django.save()
>>>flask.save()
>>>bottle.save()
>>>java = Language(name='Java')
>>>java.save()
>>>spring = Framework(name='Spring', language=java)
>>>spring.save()
```

9.1.2. Vídeo 2: Query-one-to-many

- **Shell – Descripción:** Estas consultas de Django empleadas en el video muestran cómo interactuar con modelos relacionados mediante claves foráneas y filtros. Comienza importando los modelos Language y Framework, seguido por una consulta para obtener todos los objetos del modelo Framework. Luego, se aplican filtros para encontrar frameworks según el nombre del lenguaje asociado, devolviendo aquellos relacionados con 'Python' y 'Java'. También se emplean filtros como startswith para buscar lenguajes que empiecen con una determinada cadena. Finalmente, se ejecutan consultas inversas utilizando la clave foránea para encontrar los lenguajes asociados con frameworks específicos, como 'Spring' y 'Bottle'.

```
>>> from example.models import Language, Framework
>>> Framework.objects.all()
<QuerySet [<Framework: Django>, <Framework: Flask>, <Framework: Bottle>,
    <Framework: Spring>]>
>>> Framework.objects.filter(language__name='Python')
<QuerySet [<Framework: Django>, <Framework: Flask>, <Framework: Bottle>]>
>>> Framework.objects.filter(language__name='Java')
<QuerySet [<Framework: Spring>]>
>>> Framework.objects.filter(language__name__startswith='Py')
<QuerySet [<Framework: Django>, <Framework: Flask>, <Framework: Bottle>]>
>>> Framework.objects.filter(language__name__startswith='Pa')
<QuerySet []>
>>> Language.objects.filter(framework__name='Spring')
<QuerySet [<Language: Java>]>
>>> Language.objects.filter(framework__name='Bottle')
<QuerySet [<Language: Python>]>
```

9.2. Relación muchos a muchos:

9.2.1. Vídeo 3: many-to-many-relationships (Proyecto 'model examples')

- **Modelos – Descripcion:** Estos modelos implementado en el video representan una relación de muchos a muchos entre películas y personajes. El modelo Movie tiene un campo para el nombre de la película y una función str para devolver su nombre como representación legible. Por otro lado, el modelo Character representa personajes con un nombre y una relación de muchos a muchos con películas a través de un campo movies, utilizando el campo ManyToManyField para establecer esta relación. La función str del modelo Character devuelve el nombre del personaje.

```
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
```

- **Shell – Descripción:** A continuación se utiliza los modelos Movie y Character de Django para representar relaciones de muchos a muchos entre películas y personajes. Comienza creando una película llamada .ªvengersz un personaje llamado Çaptain America", y luego asigna la película .ªvengers.ªl personaje Çaptain America". Luego, se crean más películas y personajes, y se establecen relaciones entre ellos mediante el uso de métodos como add() y create() en los campos de relación ManyToManyField. En resumen, este código ilustra cómo trabajar con relaciones de muchos a muchos en un modelo de base de datos utilizando Django.

```
>>>from example.models import Movie, Character
>>>avengers = Movie(name='Avengers')
>>>avengers.save()
>>>captain_america = Character(name='Captain America')
>>>captain_america.save()
>>>captain_america.movies.add(avengers)
>>>civil_war = Movie(name='Civil War')
>>>thor = Movie(name='Thor: Dark World')
>>>thor_character = Character(name='Thor')
>>>civil_war.save()
>>>thor.save()
>>>thor_caracter.save()
>>>captain_america.movies.add(civil_war)
>>>thor_character.movies.add(avengers)
>>>thor_character.movies.add(thor)
>>captain_america.movies.create(name='Winter Soldier')
<Movie: Winter Soldier>
```

9.2.2. Vídeo 4: many-to-many-Query

- **Shell – Descripción:** En este fragmento se ilustra consultas utilizando relaciones de muchos a muchos entre los modelos Movie (Película) y Character (Personaje). Primero se importan los modelos relevantes del módulo example.models. Luego, se ejecutan varias consultas para buscar información específica en la base de datos. Por ejemplo, se realiza una consulta para encontrar personajes asociados a la película Çivil Warz otra para encontrar películas que incluyan un personaje con el nombre Çaptain Am". Estas consultas demuestran cómo utilizar relaciones de muchos a muchos para obtener datos interrelacionados de manera eficiente en Django.

```
>>> from example.models import Movie, Character
>>> Character.objects.filter(movies__name='Civil War')
<QuerySet [<Character: Captain Am>]>
>>> Movie.objects.filter(character__name='Captain Am')
<QuerySet [<Movie: Avengers>, <Movie: Civil War>, <Movie: Winter Sol>]>
>>> captain_america = Character.objects.get(name='Captain Am')
>>> captain_america
<Character: Captain Am>
>>> captain_america.movies.all()
<QuerySet [<Movie: Avengers>, <Movie: Civil War>, <Movie: Winter Sol>]>
>>> avengers = Movie.objects.get(name='Avengers')
>>> avengers
<Movie: Avengers>
>>> avengers.character_set.all()
<QuerySet [<Character: Captain Am>, <Character: Thor>]>
```

9.3. Configuración Base de Datos (Proyecto 'model examples')

Esta configuracion se hace en settings.py, se hizo al iniciar toda la actividad ya que para los primero video se uso mySQL de xampp

9.3.1. Vídeo 5: Database-settings

- **Shell – Descripción:** Este fragmento de configuración se refiere a la configuración de la base de datos en un proyecto Django. Especifica que se está utilizando el motor MySQL para la base de datos por defecto, con el nombre de la base de datos "django", el usuario "Victor", la contraseña "maldonado100204.", el host "localhostz el puerto "3306".
  Para poder usar MySQL en un proyecto Django, es necesario tener instalado el paquete mysqlclient. Este paquete proporciona el adaptador de base de datos necesario para conectar Django con MySQL. Se puedo instalar usando pip: "`pip install mysqlclient`".

```
80  # Replica actividad del quinto Video 'DataBase Settings'.
81  # Esta actividad se hizo desde el principio debido a que se hace uso de mysql de xampp en este proyecto
82
83  DATABASES = {
84      'default': {
85          'ENGINE': 'django.db.backends.mysql',
86          'NAME': 'django',
87          'USER': 'Victor',
88          'PASSWORD': 'maldonado100204.',
89          'HOST': 'localhost',
90          'PORT': '3306',
91      }
92  }
93  # Finzalizacion de la actividad del quinto Video 'DataBase Settings'.
```

Figura 3: Configuración Base de Datos - mySQL

9.4. Impresión de pdfs (Proyecto 'pdf printing')

9.4.1. Vídeo 6: Render a Django HTML Template to a PDF file Django Utility CFE Render-to-PDF

- **VISTAS – Descripción:** Este siguiente código Django muestra cómo generar un archivo PDF a partir de una plantilla HTML. Primero, se importan las bibliotecas necesarias, se establece la configuración regional para formatear la moneda y se define una clase llamada GeneratePDF que hereda de View.
  En el método get, se carga la plantilla HTML utilizando get template y se crea un contexto con datos de ejemplo, como el nombre del cliente, el número de factura, el monto y la fecha. Luego, se renderiza la plantilla con el contexto para obtener el HTML y se utiliza la función 'render to pdf' para convertirlo en un archivo PDF.
  Si se genera correctamente el PDF, se crea una respuesta HTTP con el PDF adjunto para descargarlo o visualizarlo en línea, según el parámetro download en la URL. Si no se puede generar el PDF, se devuelve un mensaje de error.

```
from django.views.generic import View
from django.template.loader import get_template
from .renderers import render_to_pdf
import locale
locale.setlocale(locale.LC_ALL, "")

class GeneratePDF(View):
    def get(self, request, *args, **kwargs):
        template = get_template('pdf/invoice.html')
        invoice_number = "007cae"
        context = {
            "customer_name": "Ethan Hunt",
            "invoice_number": f"{invoice_number}",
            "amount": locale.currency(100_000, grouping=True),
            "date": "2021-07-04",
            "pdf_title": f"Invoice #{invoice_number}",
        }
        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" %("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')
```

- **RENDERERS – Descripción:** *(Se creo el archivo renderers.py en el proyecto)* Este codigo sirve para convertir una plantilla HTML en un archivo PDF utilizando la librería xhtml2pdf. Se importan módulos como BytesIO para manejar datos binarios, HttpResponse para enviar la respuesta con el PDF, 'get template' para cargar la plantilla HTML, y pisa para la conversión.
  La función 'render to pdf' recibe la ruta de la plantilla y un diccionario de contexto opcional. Después de cargar y renderizar la plantilla con el contexto, se utiliza pisa para crear el PDF. Si hay algún problema durante este proceso, se devuelve una respuesta HTTP con el mensaje Ïnvalid PDFz un código de estado 400.

```
# Video 6 'Render a Django HTML Template to a PDF file Django Utility CFE
    Render to PDF'
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 pdf.err:
        return HttpResponse("Invalid PDF", status_code=400,
            content_type='text/plain')
    return HttpResponse(result.getvalue(), content_type='application/pdf')
```

- **URLS – Descripción:** El siguiente código define las URL de un proyecto Django. Al acceder a la URL /pdf/, se activa la vista GeneratePDF que está definida en el archivo de vistas (views.py). Esta vista se encarga de generar un archivo PDF a partir de una plantilla HTML y enviarlo como respuesta.

```
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()),
]
```

- **Plantilla invoice.html – Descripción:** Este fragmento de código representa una plantilla HTML para generar facturas en un proyecto Django. Incluye estilos CSS para dar formato al contenido, como el tamaño y el color de la fuente, así como la alineación del texto. Utiliza variables del contexto, como 'invoice number', 'customer name', amount, y date, para personalizar la información de cada factura generada en PDF.

```
<!-- Video 6 'Render a Django HTML Template to a PDF file Django Utility CFE
    Render to PDF' -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>New Title Here</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;
            padding: 10px 20px 0px 20px;
          }
          .title span {
            color: #007cae;
          }
          .details {
            padding: 10px 20px 0px 20px;
            text-align: left !important;
          }
          .hrItem {
            border: none;
            height: 1px;
            color: #333;
            background-color: #fff;
          }
        </style>
      </head>
      <body>
        <div class='wrapper'>
          <div class='header'>
            <p class='title'>Invoice #{{ invoice_number}}</p>
          </div>
          <div class='details'>
            Bill to: {{ customer_name }}<br/>
            Amount: {{ amount }} <br/>
            Date: {{ date }}
            <hr class='hrItem' />
          </div>
        </div>
      </body>
    </html>
```

- **Ejecución:**

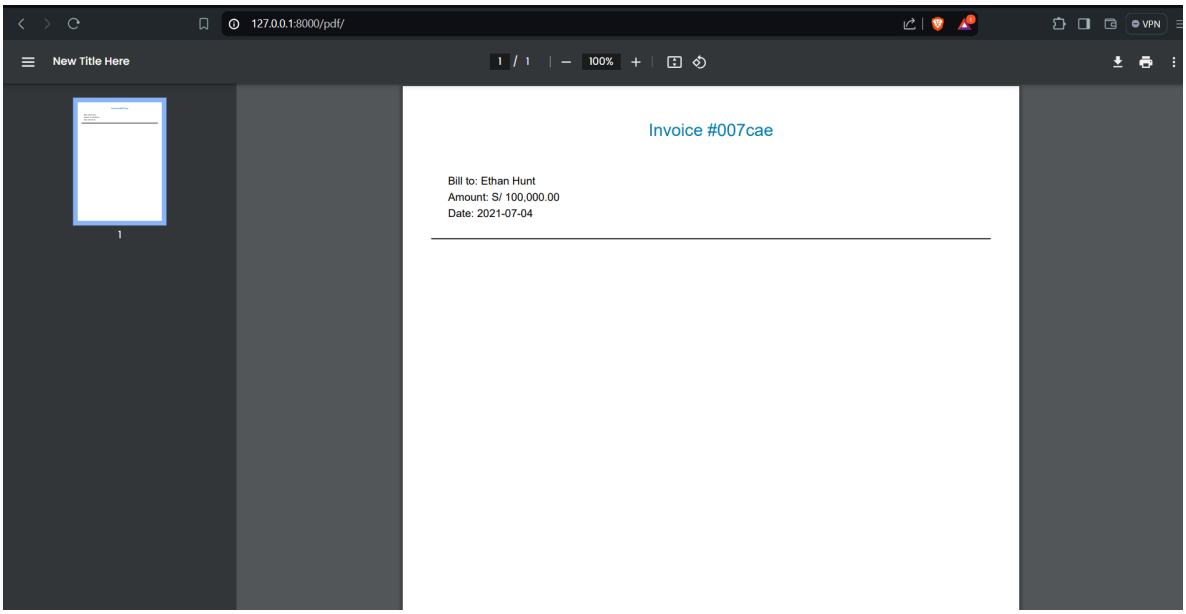

Figura 4: Ejecución

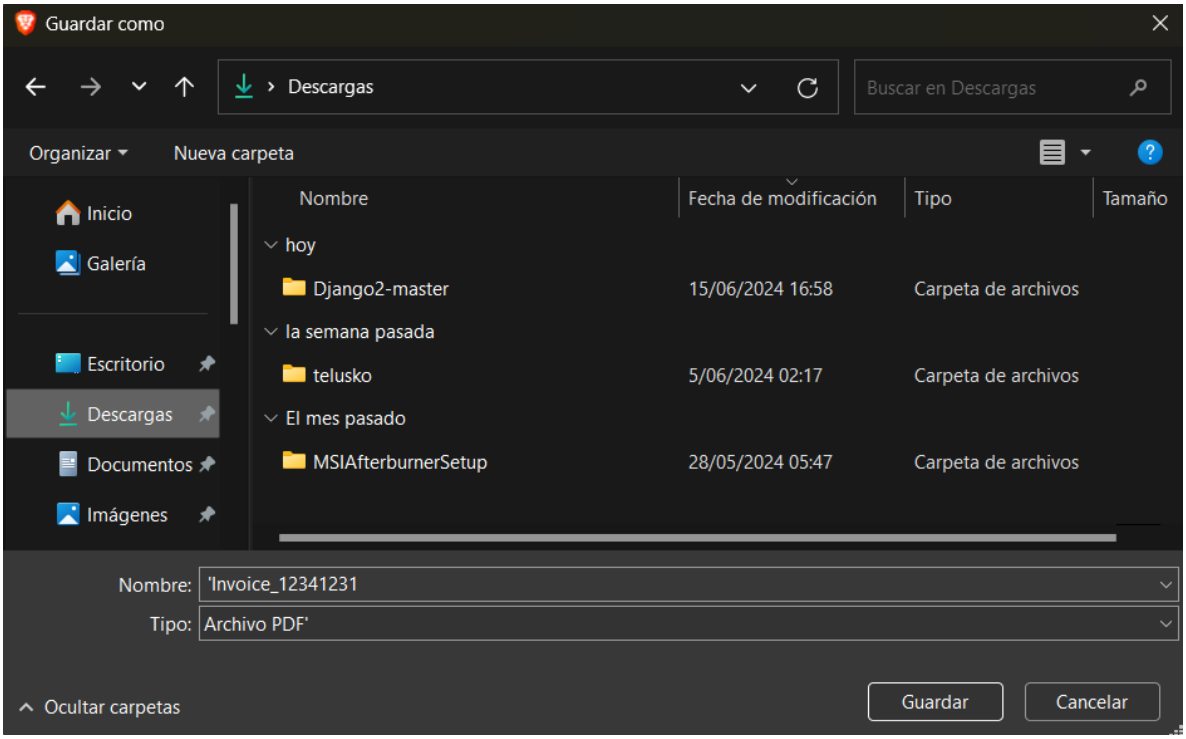

Figura 5: Ejecución

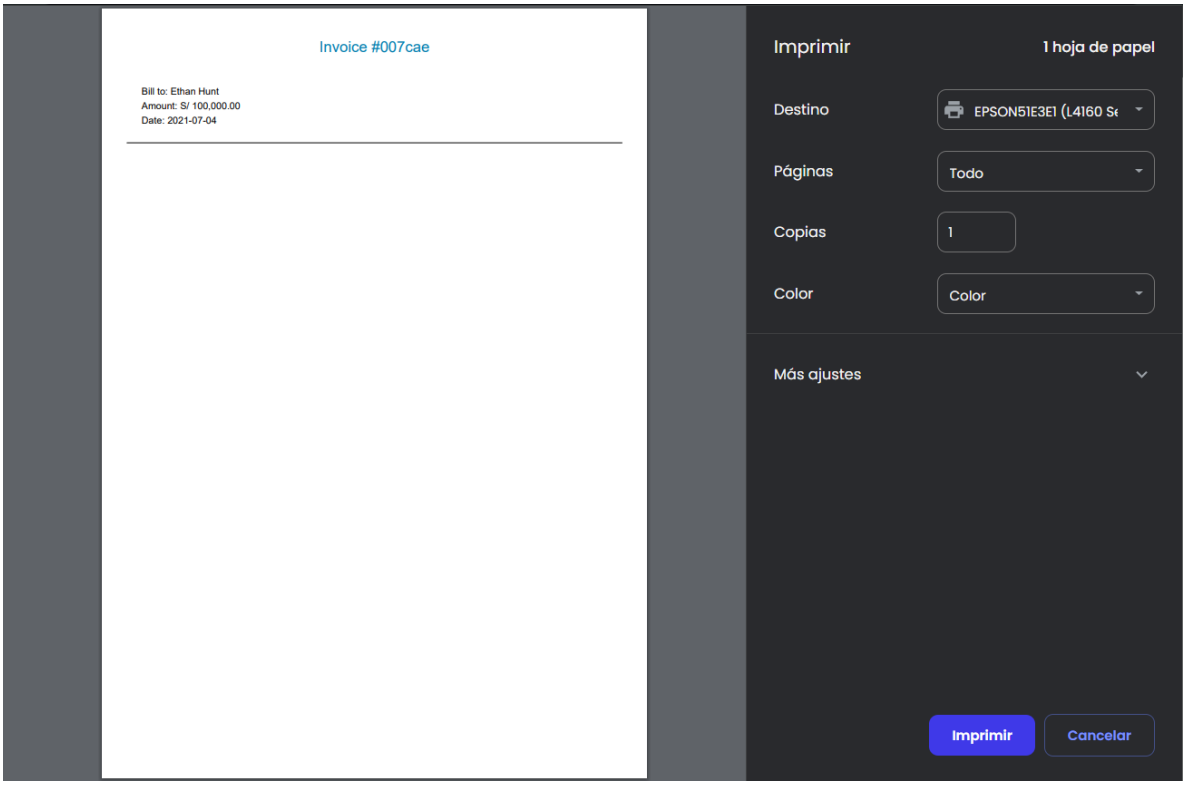

Figura 6: Ejecución

9.5. Envio de emails (Proyecto 'email project')

9.5.1. Vídeo 7: Sending Emails in Django

- **Configuración de email en setting.py:**

```
130
131  EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
132  EMAIL_HOST = 'smtp.gmail.com'
133  EMAIL_PORT = 587
134  EMAIL_USE_TLS = True
135  EMAIL_USE_SSL = False
136  EMAIL_HOST_USER = 'victorgonzalomaldonadovilca@gmail.com'
137  EMAIL_HOST_PASSWORD = ''
```

Figura 7: Configuración

- **VISTAS – Descripción:** En este fragmento de código se muestra cómo enviar un correo electrónico desde una vista. Utiliza la función 'send mail' para enviar un saludo simple ("Hola Mundo!!") desde la dirección de correo electrónico 'victorgonzalomaldonadovilca@gmail.com' a 'vmaldonadov@unsa.edu.pe'. Luego, renderiza una plantilla HTML para la página de índice ('index.html').

```
from django.shortcuts import render
```

```
from django.core.mail import send_mail

# Create your views here.

def index(request):
    send_mail(
        'Saludo desde Django',
        'Hola Mundo!!',
        'victorgonzalomaldonadovilca@gmail.com',
        ['vmaldonadov@unsa.edu.pe'],
    )
    return render(request, "email/index.html")
```

- **URLS (Project) – Descripción:** El siguiente código configura las rutas del proyecto Django. Conecta la URL 'admin/' al panel de administración de Django y la URL principal (") a las rutas definidas en 'email app.urls', permitiendo acceder al panel de administración y a las vistas de la aplicación 'email app' a través de las rutas correspondientes en esos archivos.

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

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

- **URLS (APP) – Descripción:** El código define las rutas de la aplicación Django. Enlaza la URL principal con la vista 'index' de la aplicación actual ('.'). Esto permite acceder a la vista 'index' al ingresar a la URL principal del proyecto.

```
# Video 7 'Sending Emails in Django'

from django.urls import path
from . import views

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

- **Plantilla index.html – Descripción:** Este código HTML crea una página básica que muestra un encabezado que dice "Send a Email".

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="UTF-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  </head>
  <body>
    <h1>Send a Email</h1>
  </body>
</html>
```

- **Ejecución:**

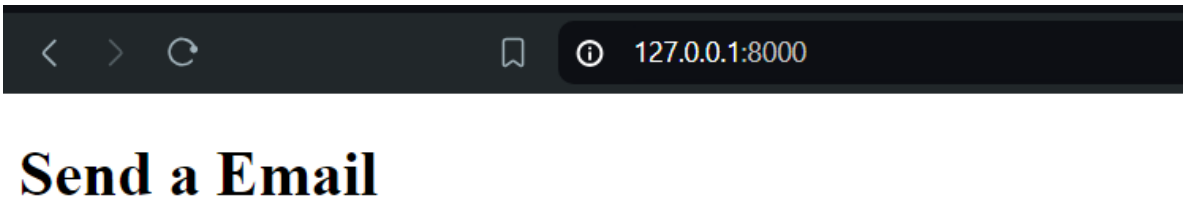

Figura 8: Ejecución

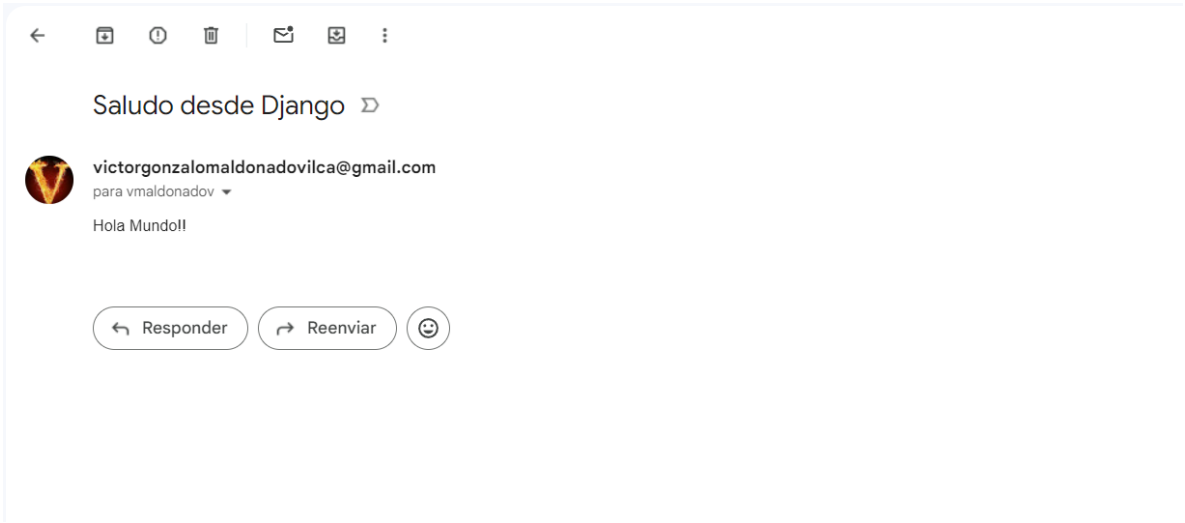

Figura 9: Ejecución - Emisor

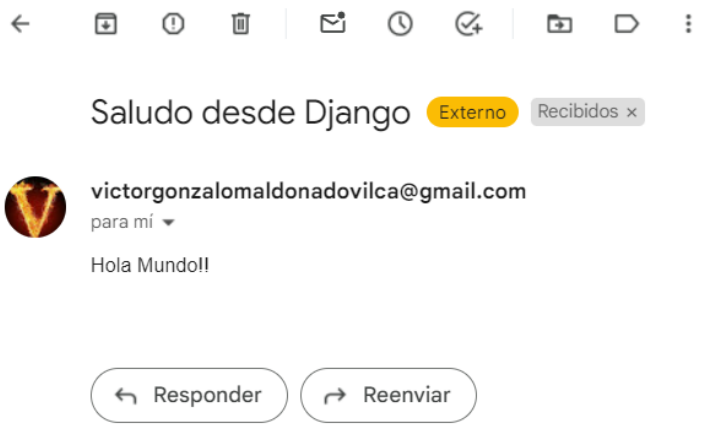

Figura 10: Ejecución - Receptor

9.6. Uso de GitHub

9.6.1. Usuario de GitHub

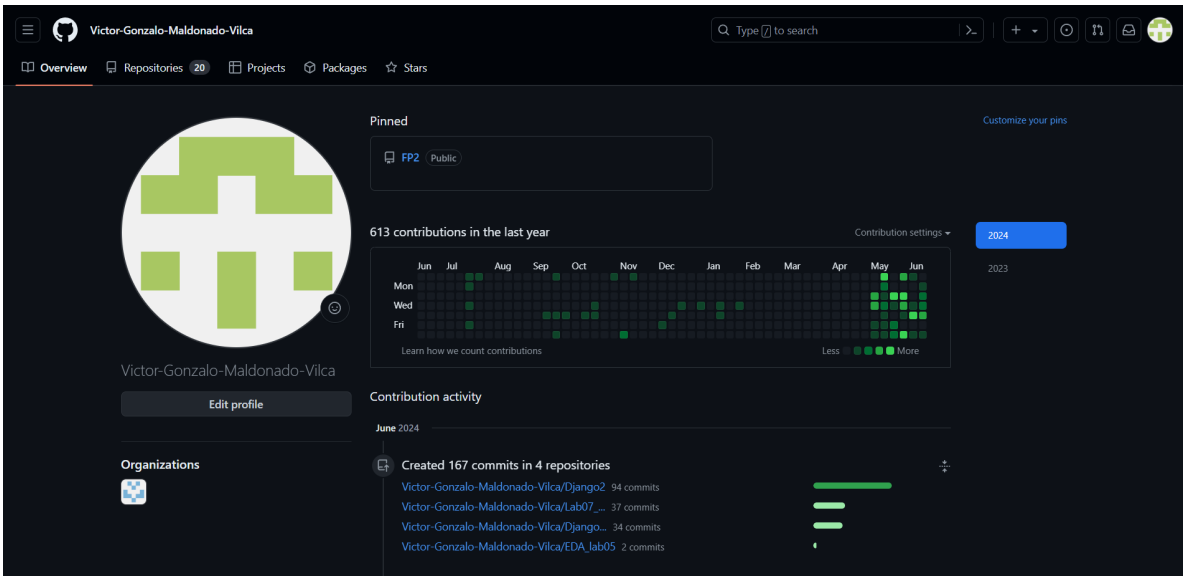

Figura 11: Usuario

9.6.2. Implementación de Readme.md

Figura 12: README.md

9.6.3. Registro de cambios en mi código

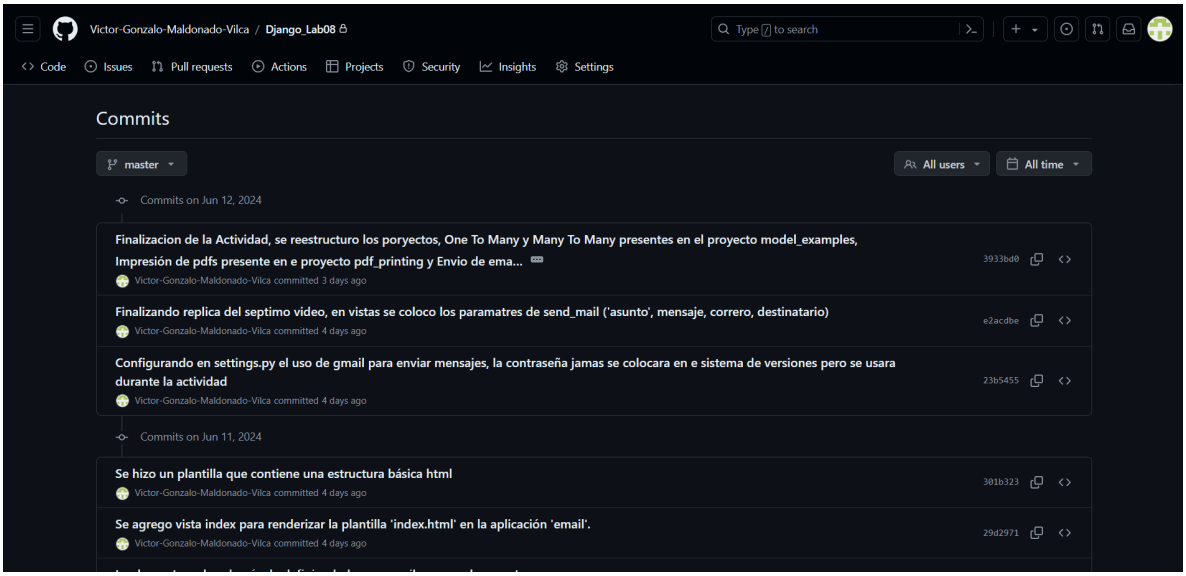

Figura 13: Commits

9.6.4. Repositorio

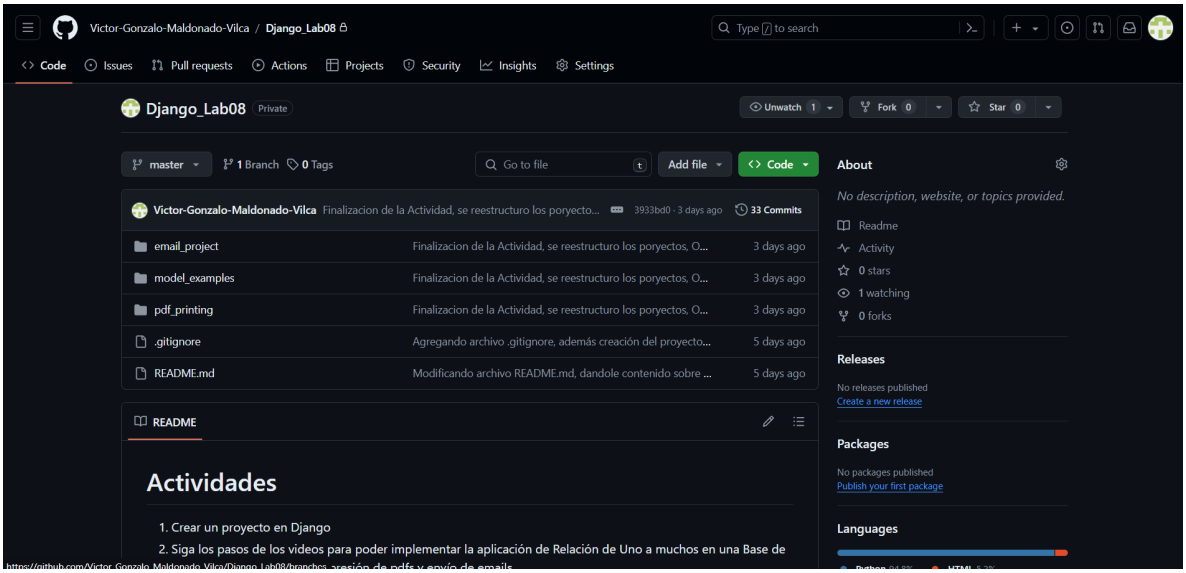

Figura 14: Repositorio

9.6.5. Proyecto compartido con el profesor de github

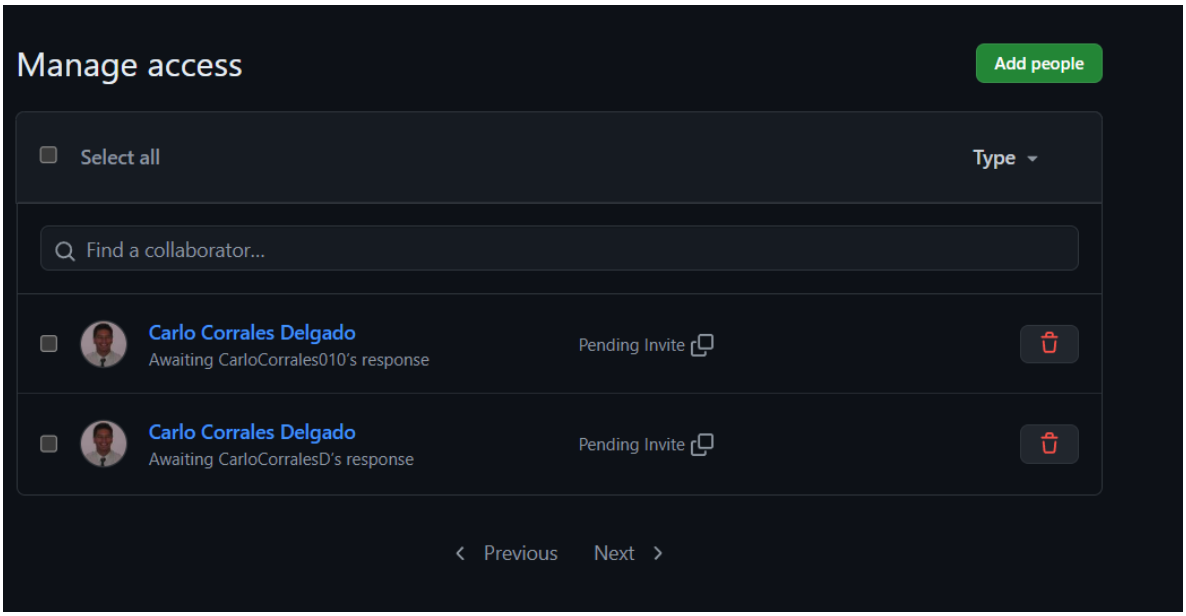

Figura 15: Compartir con el Docente

10. Recomensaciones

- Utiliza las relaciones de uno a muchos (`ForeignKey`) y muchos a muchos (`ManyToManyField`) de manera adecuada según la estructura de tus datos. Esto te permitirá organizar y gestionar la información de forma eficiente.
- Al imprimir PDFs, asegúrate de utilizar bibliotecas como xhtml2pdf o ReportLab para generar documentos de alta calidad y compatibles con diferentes dispositivos y navegadores.
- Al enviar emails, verifica que la configuración de SMTP en Django esté correctamente establecida y que los correos se envíen de manera segura para evitar problemas de entrega.

11. Conclusiones

- Las relaciones de uno a muchos y muchos a muchos son fundamentales en el diseño de bases de datos relacionales en Django, facilitando la organización y asociación de los datos entre diferentes modelos.
- La generación de PDFs y el envío de emails son funcionalidades potentes que amplían las capacidades de una aplicación Django, permitiendo la entrega de información en formatos estáticos y dinámicos.
- Es importante considerar aspectos de seguridad y rendimiento al implementar la impresión de PDFs y el envío de emails para garantizar una experiencia óptima para los usuarios finales.

11.1. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplio con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos lo items.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 1: Niveles de desempeño

	Nivel			
Puntos	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
2.0	0.5	1.0	1.5	2.0
4.0	1.0	2.0	3.0	4.0

Tabla 2: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
6. Fechas	Las fechas de modificación del código fuente estan dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	4	
Total		20		20	

12. Referencias

- https://docs.djangoproject.com/es/3.2/
- https://docs.djangoproject.com/en/5.0/
- https://docs.github.com/es
- https://git-scm.com/doc