

Informe de Laboratorio 07

INFORMACIÓN BÁSICA					
ASIGNATURA:	PROGRAMACION WEB 2				
TITULO DE LA PRÁCTICA:	Relaciones de uno a muchos, muchos a muchos y impresion de pdf y emails				
NÚMERO DE PRÁCTICA:	07	AÑO LECTIVO:	2023-A	NRO. SEMESTRE:	III
FECHA DE PRE-SENTACIÓN:	08/07/2023	HORA DE PRE-SENTACIÓN:	08/07/2023		
INTEGRANTE (s): - VLADIMIR ARTURO SULLA QUISPE				NOTA (0-20)	
GITHUB : https://github.com/Vladimir2003-debug/PW2-LAB07.git					
FLIP : https://flip.com/5f1a56f8					
DOCENTE(s): - CARLO JOSE LUIS CORRALES DELGADO					

Contents

1 TAREA	1
2 MATERIALES USADOS	2
3 RESOLUCION	2
3.1 ONE TO MANY RELATIONS	2
3.1.1 Query de uno a muchos	3
3.2 MANY TO MANY RELATIONS	4
3.2.1 Query Many To Many Relations	7
4 RENDER PDF FILE IN DJANGO	7
5 SENDING EMAILS IN DJANGO	11
5.1 POSTFIX	12
6 COMMITS	14
References	16

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
5. Crear su video Flipgrid:

2 MATERIALES USADOS

- Sistema Operativo Ubuntu, Windows 10
- Django 4.2.2
- Python 3.10.6
- pip 22.0.2
- git version 2.34.1
- virtualenv 20.13.0
- postfix software

3 RESOLUCION

3.1 ONE TO MANY RELATIONS

Con el modelo inicial que contiene Simple, DateExample y NullExample añadimos Lenguaje y Framework. Luego procedemos a hacer lo siguiente en la shell de python:

```
(env) PS C:\Users\vladimir\eclipse-workspace\PW2 Lab07\model_examples>
python .\manage.py shell
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v
.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from apps.examples.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')
>>> spring = Framework(name='Spring', language = java)
>>> java.save()
>>> spring.save()
```

Listing 1: Shell de Python

En el código anterior vemos cómo crear varios objetos language, para posteriormente crear objetos framework que al momento de crearlos en la parte de language tenemos que seleccionar uno de los lenguajes ya creados. Esto se ve en la imagen 1.

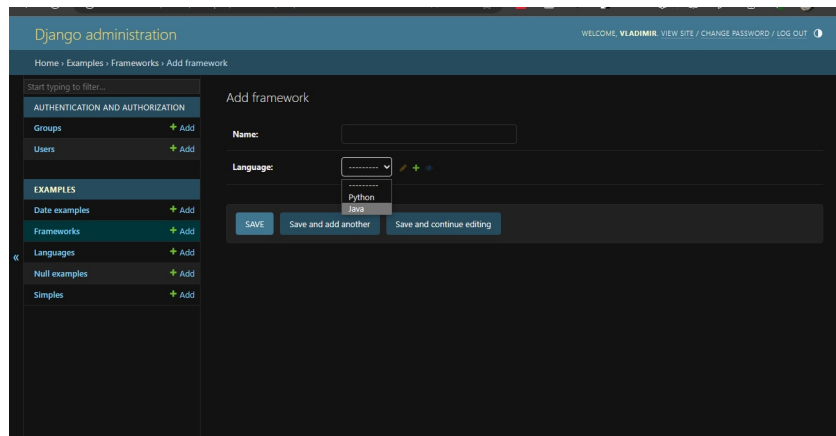


Figure 1: Resultado de crear los lenguajes python y java y su vista en el modelo Framework

3.1.1 Query de uno a muchos

Se añaden str a los modelos Language y Framework

```
class Language(models.Model):
    name = models.CharField(max_length=10)
    # Se establece que al imprimir la clase muestre name
    def __str__(self):
        return self.name
class Framework(models.Model):
    name = models.CharField(max_length=10)
    language = models.ForeignKey(Language, on_delete=models.CASCADE)
    # Al igual que en language se usa name
    def __str__(self):
        return self.name
```

Listing 2: definir str

Luego al abrir el shell de python

```
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v
.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from apps.examples.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>]>
```

3.2 MANY TO MANY RELATIONS

En esta ocasion se crearan los modelos Movie y Character ya que un Character puede pertenecer a varias peliculas y una pelicula puede tener varios actores

```
## Many to Many relationship example
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)

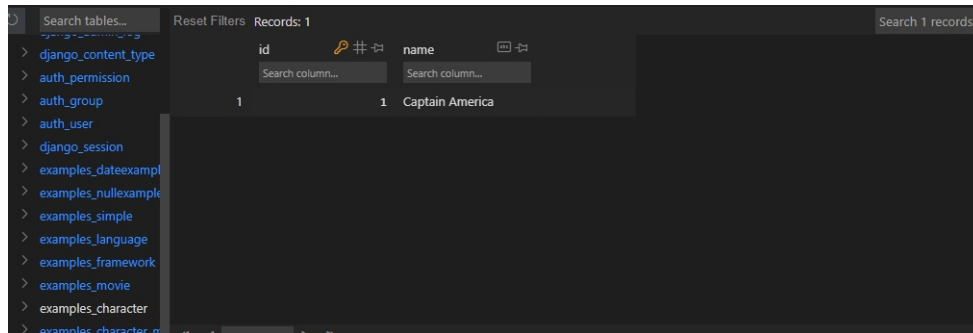
    def __str__(self):
        return self.name
```

Listing 3: Modelos de ejemplo de ManyToManyRelations

Luego se procede a indagar en el shell de python

```
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v
.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from apps.examples.models import Movie, Character
>>> avengers = Movie(name='Avengers')
>>> avengers.save()
>>> captain_america = Character(name='Captain America')
>>> captain_america.save()
```

Al crear captain america a diferencia de uno a muchos

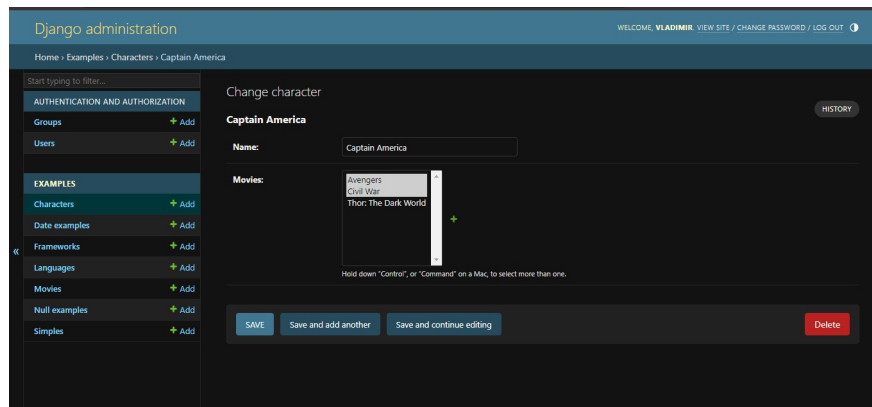


id	name
1	Captain America

Figure 2: Resultado de crear el Character captain america y su vista en la base de datos

```
>>> captain_america.movies.add(avengers)
>>> civil_war = Movie(name='Civil War')
>>> thor = Movie(name='Thor: The Dark World')
>>> thor_character = Character(name='Thor')
>>> civil_war.save()
>>> thor.save()
>>> thor_character.save()
>>> captain_america.movies.add(civil_war)
>>> thor_character.movies.add(avengers)
>>> thor_character.movies.add(thor)
```

Entonces al añadir las películas a sus respectivos actores se muestra en el admin



Django administration

Home > Examples > Characters > Captain America

Start typing to filter...

Change character

Captain America

Name: Captain America

Movies: Avengers, Civil War, Thor: The Dark World

Hold down "Control", or "Command" on a Mac, to select more than one.

SAVE Save and add another Save and continue editing Delete

Figure 3: Resultado de añadir las películas avengers y civil war

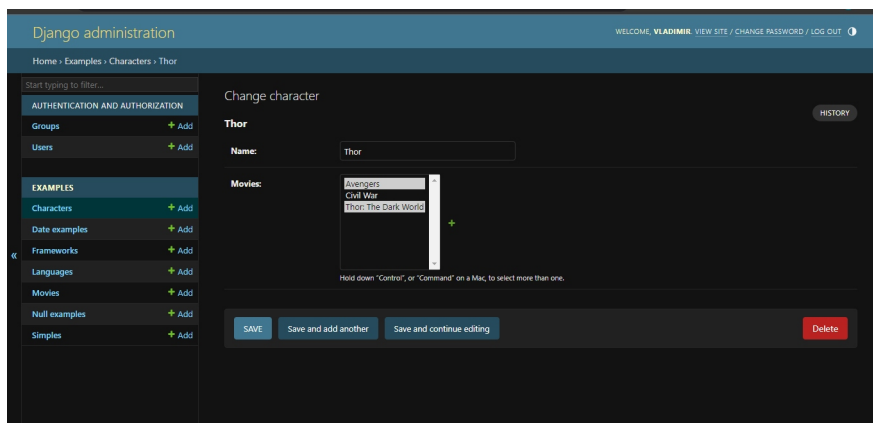


Figure 4: Resultado de añadir las películas avengers y thor: the dark world

Cabe recalcar que para que la en la tabla caracter no aparece las películas

Search tables... Reset Filters Records: 2 Search 2 records...

id	name
1	Captain America
2	Thor

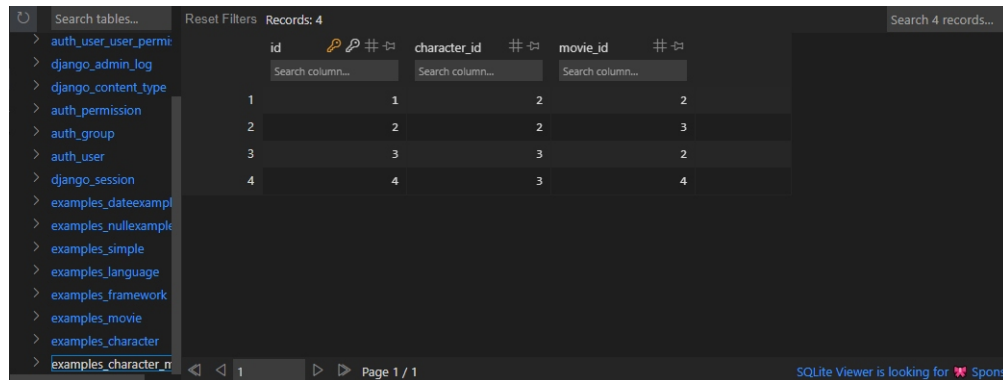
Figure 5: Tabla Character

model_examples > db.sqlite3 Search tables... Reset Filters Records: 3 Search 3 records...

id	name
1	Avengers
2	Civil War
3	Thor: The Dark World

Figure 6: tabla Movies

Entonces con el id la tabla character y movies se conectan en otra tabla



id	character_id	movie_id
1	1	2
2	2	3
3	3	2
4	4	3

Figure 7: La tabla que conecta la tabla character y movies vease los id en las tres tablas

Tambien se puede crear nuevas peliculas usando a los Characters

```
>>> captain_america.movies.create(name='Winter Soldier')
<Movie: Winter Soldier>
```

Y esta pelicula ya esta añadida al character captain_america

3.2.1 Query Many To Many Relations

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

4 RENDER PDF FILE IN DJANGO

Siguiendo las instrucciones de [1] o [2] pero con algunas modificaciones

- Vamos a crear la aplicacion render_{pdf} *elcssyjavascriptvaaestarenunarchivoaparte* primero como dicen el tutorial se instala por medio de pip

```
pip install --pre xhtml2pdf
```

Se nos instalara un monton de paquetes.

Luego se procede a crear la aplicacion render_{pdf} *contodoysustemplatesdespuesdeconfigurarlostemplatesensettings*

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [os.path.join(BASE_DIR, 'templates')],  
        'APP_DIRS': True,  
        'OPTIONS': {  
            'context_processors': [  
                'django.template.context_processors.debug',  
                'django.template.context_processors.request',  
                'django.contrib.auth.context_processors.auth',  
                'django.contrib.messages.context_processors.messages',  
            ],  
        },  
    },  
]
```

Listing 4: cambiar la configuracion de los templates

```
STATIC_URL = 'static/'  
STATICFILES_DIRS = [  
    os.path.join(BASE_DIR, 'static')  
]  
STATIC_ROOT = os.path.join(BASE_DIR, 'assets')
```

Listing 5: (opcional) configurar archivos static y assets

```
mkdir apps  
cd apps  
django-admin startapp render_pdf
```

Listing 6: instalacion de render_{pdf}

Una vez terminado procedemos a crear un archivo llamado `utils` que esta en la app y tambien creamos el archivo `invoice.html` en `templates/pdf`

```
model_examples/  
|--- apps  
|   |--- examples  
|   |--- render_pdf  
|       |--- migrations  
|       |--- static  
|           |--- css  
|           |--- js  
|       |--- templates  
|           |--- pdf  
|               |--- invoice.html  
|--- __init__.py  
|--- admin.py  
|--- apps.py  
|--- models.py  
|--- tests.py  
|--- utils.py  
|--- views.py
```



```
|--- model_examples
|--- __pycache__
|--- asgi.py
|--- settings.py
|--- urls.py
|--- wsgi.py
```

en utils.py añadimos el siguiente código

```
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 invoice se añaden las variables invoice.id, customer_name y today para ver si se imprimen las variables en el pdf

```
<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>
```

En views añadimos el siguiente código

```
from django.http import HttpResponse
from django.views.generic import View
from django.template.loader import get_template
from .utils import render_to_pdf #created in step 4

class GeneratePdf(View):
    def get(self, request, *args, **kwargs):
        template = get_template('pdf/invoice.html')
        context = {
            'invoice_id': 123,
```

```
        'customer_name': 'Vladimir Sulla',
        'amount': 12000,
        'today': "Today",
    }
    pdf = render_to_pdf('pdf/invoice.html', context)
    return HttpResponse(pdf, content_type='application/pdf')
```

finalmente en url se añade el enlace

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

urlpatterns = [
    path('', GeneratePdf.as_view(), name="pdf"),
]
```

y en urls de proyecto

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

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

Esta version solamente muestra el pdf. Para hacer un enlace que automaticamente descargue el pdf se añade lo siguiente

```
from django.http import HttpResponse
from django.views.generic import View
from django.template.loader import get_template
from .utils import render_to_pdf #created in step 4

class GeneratePdf(View):
    def get(self, request, *args, **kwargs):
        template = get_template('pdf/invoice.html')
        context = {
            'invoice_id': 123,
            'customer_name': 'Vladimir Sulla',
            'amount': 12000,
            'today': "Today",
        }
        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 content
return HttpResponse('No Found')
```

Listing 7: funcion modificada para un enlace que descargue automaticamente

5 SENDING EMAILS IN DJANGO

Siguiendo las instrucciones en <https://www.youtube.com/watch?v=X7DWErkNVJs> Creamos una pagina temporal en <https://temp-mail.org/>

Procedemos a crear un email temporal la pagina <https://temp-mail.org/>

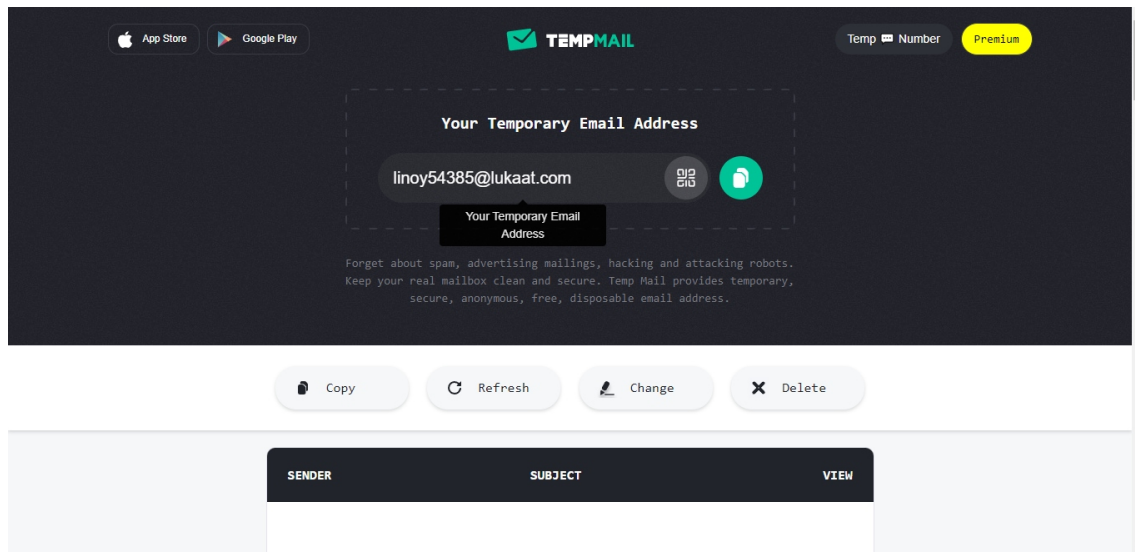


Figure 8: Pagina para crear un email temporal

Luego en views creamos por el momento un metodo para ver index.html que esta en templates/send/

```
from django.shortcuts import render

# Create your views here.

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

Creamos una platilla simple en index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>HTML 5 Boilerplate</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
```

```
<h1>Send a email</h1>
</body>
</html>
```

El tutorial usa un django 2.0 que esta desactualizado pero en [3] esta la forma de enviar email mas actualizado

Entonces en view se cambia index por el siguiente codigo

```
from django.core.mail import send_mail

send_mail(
    "Subject here",
    "Here is the message.",
    "vladimir@gmail.com",
    ["to@example.com"],
    fail_silently=False,
)
```

Finalmente dejamos la parte mas dificil para el final, en settings.py del proyecto se procede a poner este codigo

```
EMAIL_HOST = 'smtp.hushmail.com'
EMAIL_PORT = 578
EMAIL_HOST_USER = 'anthony@prettyprinted.com'
EMAIL_HOST_PASSWORD = ''
EMAIL_USE_TLS = True
EMAIL_USE_SSL = False
```

Sin embargo hushmail el cual es un software para el envio de correo es de pago y no esta disponible. De hecho se presentaron varios problemas en la autenticacion y en la forma en la que el destinatario recibia el correo sin embargo hay otra manera de hacerlo sin necesidad de crear cuenta. Cabe resaltar que esto se hizo en un SO Ubuntu dentro de una maquina virtual.

5.1 POSTFIX

Segun [6] y [5] postfix es de codigo abierto y alternativa a sendmail que permite usar el localhost para envio de mails. Entonces se procede con la instalacion

```
sudo apt-get install postfix
```

En medio de la instalacion se nos muestra lo siguiente:

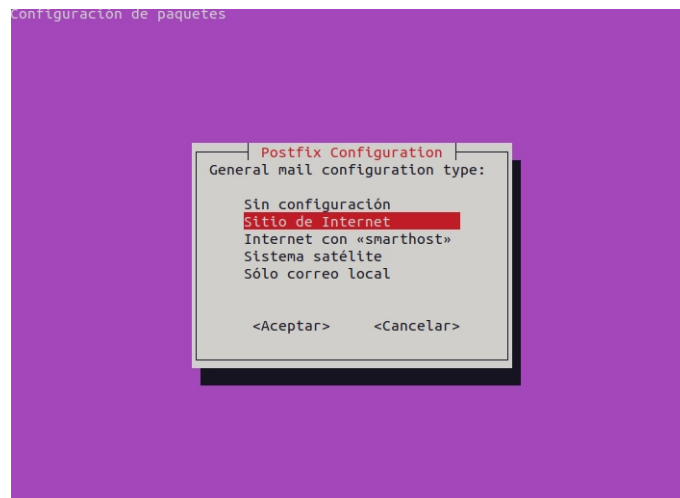


Figure 9:

En donde nos indica el tipo de configuracion, Se selecciona "sitio de internet"

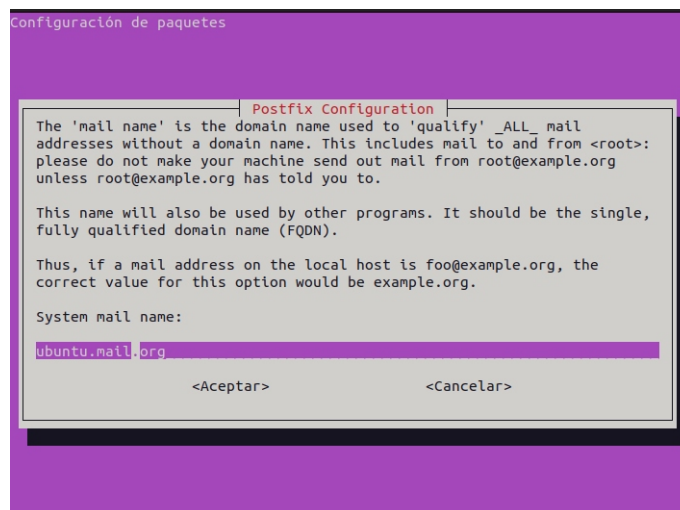


Figure 10:

Luego de aceptar en la pagina anterior nos aparecera esta pantalla. Cambiamos el nombre si lo requerimos

Entoonces luego de hacer estos pasos procedemos con la configuracion en settings.py en el proyecto.

```
# A partir de django 1.6+ se requiere poner esta linea de codigo
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
# Como se usa postfix se pude usar el local host para enviar correo
EMAIL_HOST = 'localhost'
# El port por defecto para localhost vease la documentacion
# de django EMAIL_PORT
EMAIL_PORT = 25
EMAIL_HOST_USER = ''
EMAIL_HOST_PASSWORD = ''
```

Para evitar la autenticacion lo mas posible

EMAIL_USE_TLS = False

EMAIL_USE_SSL = False

ACCOUNT_EMAIL_VERIFICATION = 'none'

La informacion fue adquirida principalmente de [4]

Una vez cambiado ya se puede enviar correos

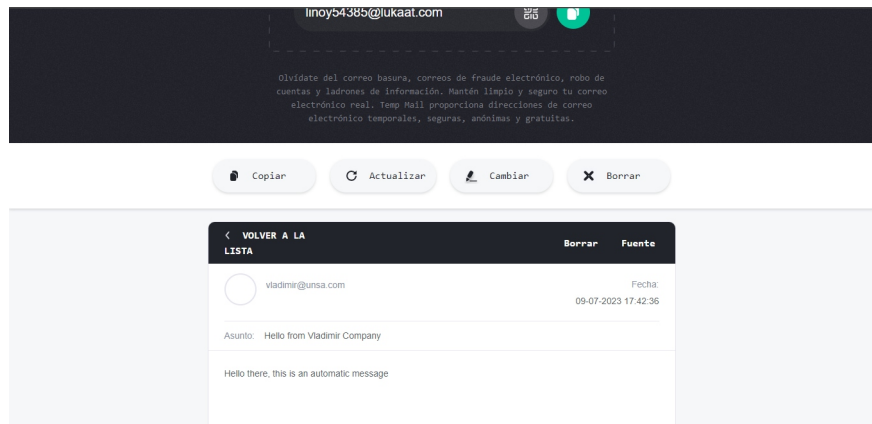


Figure 11: El email temporal ya puede recibir correos

6 COMMITS

los ultimos commits son los siguientes:

```
(env) $git log
commit 06ba38097adb15bca9a81f4367e86d983562ceb4 (HEAD -> main, origin/
main)
Author: Vladimir Arturo Sulla Quispe <vsullaq@unsa.edu.pe>
Date: Sun Jul 9 21:18:10 2023 -0500

    a adido comentarios a los settings de email

commit 0f48a908a3a7f7bfc5f7f9f44dd8e5a95bcd51a2
Author: alumno <alumno@ubuntu.mail.org>
Date: Sun Jul 9 17:45:10 2023 -0500

    modificado para envio de email usando postfix

commit 2f713ef95b0bc1c3790e5e5b1f633d64c0a52e54
Author: alumno <alumno@ubuntu.mail.org>
Date: Sun Jul 9 17:44:19 2023 -0500

    modificado el email que envia

commit 03289cf4d78fd34f0c0e3284cdbe762742b9e5e2
Author: Vladimir Arturo Sulla Quispe <vsullaq@unsa.edu.pe>
Date: Sat Jul 8 23:14:54 2023 -0500
```

a adido urls

```
commit c23448a302d920544a1b8c4563f540f0143bd788
Author: Vladimir Arturo Sulla Quispe <vsullaq@unsa.edu.pe>
Date: Sat Jul 8 23:14:35 2023 -0500
```

creado templates

```
commit 89fa641913feef1a4dbc59dd97e95927902f2f34
Author: Vladimir Arturo Sulla Quispe <vsullaq@unsa.edu.pe>
Date: Sat Jul 8 23:14:09 2023 -0500
```

a adido enalace send

```
commit 2dc298d8ab623626973919f7551151846e1b33c5
Author: Vladimir Arturo Sulla Quispe <vsullaq@unsa.edu.pe>
Date: Sat Jul 8 23:13:52 2023 -0500
```

tratando de que se envíe correos por django

```
commit b8041696c4ebfbaf9c0acf820aaf35a7164108e9
Author: Vladimir Arturo Sulla Quispe <vsullaq@unsa.edu.pe>
Date: Sat Jul 8 23:13:30 2023 -0500
```

cambiado la forma en la que enviar send email

```
commit b179b8bcc02ab5386b1d91865e9497023fd9eacb
Author: Vladimir Arturo Sulla Quispe <vsullaq@unsa.edu.pe>
Date: Sat Jul 8 23:12:59 2023 -0500
```

cambiado nombre de apps en send

```
commit 9b951b4d027224a0c4a14ac9d7d89f2037507997
Author: Vladimir Arturo Sulla Quispe <vsullaq@unsa.edu.pe>
Date: Sat Jul 8 23:12:36 2023 -0500
```

cambiado conpre de apps en exampes

```
commit 1260f215c0774d244e477cf413a2e5931eb1283b
Author: Vladimir Arturo Sulla Quispe <vsullaq@unsa.edu.pe>
Date: Sat Jul 8 14:16:08 2023 -0500
```

registrado app en settings

```
commit e402f859f5e2829f09c48d9fd6e294b5add5858e
Author: Vladimir Arturo Sulla Quispe <vsullaq@unsa.edu.pe>
Date: Sat Jul 8 14:15:42 2023 -0500
```

cambiado nombre de la app send

```
commit d9a96f3fce768b84e6067dc30a7ab8f329c1fefc
```

Author: Vladimir Arturo Sulla Quispe <vsullaq@unsa.edu.pe>

Date: Sat Jul 8 14:14:42 2023 -0500

creada la aplicacion send

commit 0d6045c3174440d101b2f3fe7fb1d7fb32d17edb

Author: Vladimir Arturo Sulla Quispe <vsullaq@unsa.edu.pe>

Date: Sat Jul 8 13:34:44 2023 -0500

cambiado content a response, se elimino comillas simples a la
descarga de invoice

commit b45c80bd8779fa33a8f01866f303dc68c9b6b4a3

Author: Vladimir Arturo Sulla Quispe <vsullaq@unsa.edu.pe>

Date: Sat Jul 8 13:34:08 2023 -0500

References

- [1]• CodingEntrepreneurs. *HTML Template to PDF in Django*. URL: <https://www.codingforentrepreneurs.com/blog/html-template-to-pdf-in-django/>. (accessed: 08.07.2023).
- [2] CodingEntrepreneurs. *Render a Django HTML Template to a PDF file // Django Utility // CFE // Render to PDF*. URL: <https://www.youtube.com/watch?v=B7EIK9yVtGY>. (accessed: 08.07.2023).
- [3] Django Documentation. *Sending email*. URL: <https://docs.djangoproject.com/en/4.2/topics/email/>. (accessed: 08.07.2023).
- [4] Ajay Gupta. *Django - [Errno 111] Connection refused*. URL: <https://stackoverflow.com/questions/5802189/django-errno-111-connection-refused>. (accessed: 09.07.2023).
- [5] POSTFIX. *Postfix*. URL: <https://es.wikipedia.org/wiki/Postfix>. (accessed: 09.07.2023).
- [6] POSTFIX. *The Postfix Home Page*. URL: <https://www.postfix.org/>. (accessed: 09.07.2023).