

## **Índice**

- 1.-Actualización Pip, Instalación/Uso Venv y Django, Creación del proyecto y Startapp. Pág.2**
- 2.-Migración, Creación de superuser y Modelos. Pág3**
- 3.- Creación de Index, base y Vistas de los modelos. Pág4-7**
- 4.-Extensión de usuario, Login/Logout,Register. Pág8-11**
- 5.-Uso de ficheros estáticos.Pág12**
- 6.-Permisos y acceso a vistas.Pág13-15**
- 7.-Añadir estilo mediante Bootstrap, Search y arreglos.Pág16**
- 8.- Api.Pág17-18**
- 9.- Alojamiento en AWS con dominio y nginx y uwsgi.Pág19-21**
- 10,-CertBot.Pág22**
- 11,-Levantar la aplicación con nginx/wsgi y certbot.Pág23**
- 12,-Uso de GBD con Postgres para la aplicación.Pág24-25**
- 13,-Uso de RDS para montar la base de datos postgres en AWS.Pág26-27**
- 14,-Proyecto CMS, creación con Wagtail y creación del grupo bloggers.Pág28-30**
- 15,-Autenticación mediante uso de LDAP.Pág31**
- 16, Script de automatización agregar usuarios al grupo Blogger.Pág32**
- 17,Script de automatización para mandar correo al usuario al agregarse a grupo Blogger.Pág33**
- 18,-Lighthouse,Http2,Compresion y Comparaciones.Pág34-35**
- 19,-Matomo.Pág36-37**

Primero probamos a actualizar pip3 de necesitarlo.

```
sacopapa@sacopapa-HP-Laptop-15-da0xxx:~/Escritorio1/Escritorio/AreaTrabajo/Nuevo_Area/EntornoVirtual$ python3 -m pip install --upgrade pip
```

Instalamos virtualenv con el uso de pip3.

```
sacopapa@sacopapa-HP-Laptop-15-da0xxx:~/Escritorio1/Escritorio/AreaTrabajo/Nuevo_Area/EntornoVirtual$ pip3 install virtualenv
```

Creamos el entorno virtual en el que vamos a trabajar.

```
sacopapa@sacopapa-HP-Laptop-15-da0xxx:~/Escritorio1/Escritorio/AreaTrabajo/Nuevo_Area/EntornoVirtual$ python3 -m venv ilmp-env
```

Accedemos al entorno virtual creado.

```
sacopapa@sacopapa-HP-Laptop-15-da0xxx:~/Escritorio1/Escritorio/AreaTrabajo/Nuevo_Area/EntornoVirtual$ source ilmp-env/bin/activate
```

Instalamos Django.

```
(ilmp-env) sacopapa@sacopapa-HP-Laptop-15-da0xxx:~/Escritorio1/Escritorio/AreaTrabajo/Nuevo_Area/EntornoVirtual$ python3 -m pip install Django
```

Creamos nuestro proyecto.

```
(ilmp-env) sacopapa@sacopapa-HP-Laptop-15-da0xxx:~/Escritorio1/Escritorio/AreaTrabajo/Nuevo_Area/AplicacionWeb$ django-admin startproject ilmp
```

Levantamos el proyecto y vemos cómo funciona y podemos acceder a él mediante el uso de un navegador a nuestra dirección de loopback puerto 8000.

```
(ilmp-env) sacopapa@sacopapa-HP-Laptop-15-da0xxx:~/Escritorio1/Escritorio/AreaTrabajo/Nuevo_Area/AplicacionWeb$ python3 ilmp/manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
```



The install worked successfully! Congratulations!

Realizamos un startapp de ilmp.

```
(ilmp-env) sacopapa@sacopapa-HP-Laptop-15-da0xxx:~/Escritorio1/Escritorio/AreaTrabajo/Nuevo_Area/AplicacionWeb/ilmp$ python3 manage.py startapp ilmp_app
(ilmp-env) sacopapa@sacopapa-HP-Laptop-15-da0xxx:~/Escritorio1/Escritorio/AreaTrabajo/Nuevo_Area/AplicacionWeb/ilmp$ ls
db.sqlite3  ilmp  ilmp_app  manage.py
```

Realizamos un migrate.

```
(ilmp-env) sacopapa@sacopapa-HP-Laptop-15-da0xxx:~/Escritorio1/Escritorio/AreaTrabajo/Nuevo_Area/AplicacionWeb/ilmp$ python3 manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001 initial... OK
```

Hecha la migración creamos un superusuario.

```
(ilmp-env) sacopapa@sacopapa-HP-Laptop-15-da0xxx:~/Escritorio1/Escritorio/AreaTrabajo/Nuevo_Area/AplicacionWeb/ilmp$ python3 manage.py createsuperuser
Username (leave blank to use 'sacopapa'): admin
```

Añadimos los modelos en models.py de ilmp\_app.

```
ilmp_app > models.py > ...
1  from django.db import models
2
3  GENDER_CHOICES = (
4      ('M', 'Male'),
5      ('F', 'Female'),
6  )
```

Creamos Gender\_choices para poder escoger género.

Añadimos las bases del modelo de mascota, perdidos y encuentros que acabaremos completando y modificando más adelante con el uso de makemigrations.

```
class Mascotas(models.Model):
    namePet = models.CharField(max_length=200)
    infoPet = models.CharField(max_length=200)
    agePet = models.DateField(blank=True, null=True)
    typePet = models.CharField(max_length=200)
    imgPet = models.ImageField(null=True, blank=True, upload_to="images/")
    genderPet = models.CharField(max_length=1, choices=GENDER_CHOICES, blank=True)
    usrPet = models.ForeignKey("User", on_delete=models.CASCADE, null=True)

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

```
class Perdidos(models.Model):
    infoLost = models.CharField(max_length=200)
    dateLost = models.DateTimeField(blank=True, null=True)
    petLost = models.ForeignKey("Mascotas", on_delete=models.CASCADE, null=True)
    ubiLost = models.CharField(max_length=200, blank=True)

    def __str__(self):
        return self.infoLost

class Encuentros(models.Model):
    typeFind = models.CharField(max_length=200)
    imgFind = models.ImageField(null=True, blank=True, upload_to="images/")
    infoFind = models.CharField(max_length=200)
    genderFind = models.CharField(max_length=1, choices=GENDER_CHOICES, blank=True)
    ubiFind = models.CharField(max_length=200, blank=True)

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

Activamos los modelos dirigiéndonos al fichero settings.py de ilmp y añadiendo su ruta.

```
INSTALLED_APPS = [
    'ilmp_app.apps.IlmpAppConfig',
    'django.contrib.admin',
    'django.contrib.auth',
```

Añadimos las vistas iniciales, para el index añadimos en views.

```
ilmp_app > views.py > index
1  from django.shortcuts import render
2  from django.http import HttpResponse
3
4  #Página de inicio
5  def index(request):
6      return HttpResponse("Index")
```

Creamos un fichero urls dentro de ilmp\_app en el que añadimos

```
from django.urls import path
#from django.conf.urls import include
from ilmp_app.views import index

urlpatterns = [
    path(r'', index),
]
```

Modificamos el fichero urls de ilmp y añadimos

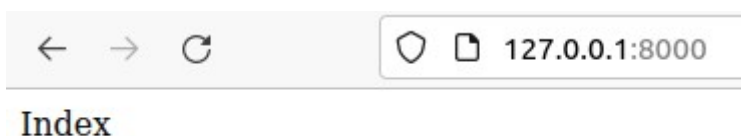
```
from django.conf.urls import include
from django.contrib import admin
from django.urls import path

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

Instalamos Pillow para que podamos utilizar las imágenes de los modelos.

```
^C(ilmp-env) sacopapa@sacopapa-HP-Laptop-15-da0xxx:~/Escritorio1/Escritorio/Area
Trabajo/Nuevo_Area/AplicacionWeb/ilmp$ python3 -m pip install Pillow
```

Realizamos makemigrations y migration y levantamos para verificar que se muestre el index inicial.



Indicamos a Django donde debe buscar nuestras plantillas añadiendo lo siguiente en dirs de settings.py.

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / "templates"],
        'APP_DIRS': True,
```



Creamos un fichero base.html que usaremos como plantilla base y la situaremos en la ubicación templates.

```
ilmp_app > templates > <> base.html > html > head
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>ILMP</title>
5      </head>
6
7      <body>
8          {% block content %}{% endblock content %}
9      </body>
10 </html>
```

Creamos ahora el index.html y añadimos la extensión del base.

```
ilmp_app > templates > <> index.html
1  {% extends 'base.html' %}
```

Modificamos la ruta de index del views.py

```
4  #Página de inicio
5  def index(request):
6      return render(request, "index.html")
```

Creamos las Listas/Detalle y vistas de creación, actualización y borrado.

```
ilmp_app > templates > mascotas_list.html
1  {% extends 'base.html' %}
2  {% load static %}
3
4
5  {% block content %}
6      <h2>Mascotas</h2>
7      <ul>
8          {% for mascota in object_list %}
9              <li><a href="{% url 'mascotas-detail' mascota.id %}">{{ mascota.namePet }}</a>
10                 <!--<li>{{ mascota.namePet }}</li> -->
11                 <a href="{% url 'mascotas-update' mascota.id %}"> <button>Modificar</button></a>
12                 <a href="{% url 'mascotas-delete' mascota.id %}"> <button>Eliminar</button></a>
13             </li>
14          {% endfor %}
15      <p><a href="{% url 'mascotas-add' %}"> <button>Añadir Mascota</button></a></p>
16      </ul>
17  {% endblock %}
```

```
ilmp_app > templates > <> mascotas_update_form.html > ...
1
2  <form enctype="multipart/form-data" method="post">{% csrf_token %}
3      {{ form.as_p }}
4      <input type="submit" value="Modificar">
5  </form>
```

```
ilmp_app > templates > <> mascotas_form.html > ...
1
2  <form enctype="multipart/form-data" method="post">{% csrf_token %}
3      {{ form.as_p }}
4      <input type="submit" value="Guardar">
5  </form>
```

```
ilmp_app > templates > <> mascotas_detail.html > ...
1  {% extends 'base.html' %}
2
3  {% block content %}
4
5      <h2>{{ object.namePet }}</h2>
6
7      <h3>{{ object.infoPet }}</h3>
8
9      </br>
10
11      <p>Género : {{ object.genderPet }}</p>
12      <p>Tipo de Animal : {{ object.typePet }}</p>
13      <p>Nacido el {{ object.agePet }}</p>
14      </br>
15      <p>Imagen : </p>
16      </br>
17      <p>Dueño : {{ object.usrPet }}</p>
18
19  {% endblock %}
```

```
ilmp_app > templates > <> mascotas_confirm_delete.html > form
1  <form method="post">{% csrf_token %}
2      <p>¿Estás seguro de eliminar la mascota? "{{ object }}"?</p>
3      <input type="submit" value="Eliminar">
4  </form>
```

Realizamos formularios similares para Perdidos y Encuentros.

Importamos y añadimos en urls.py

```
from ilmp_app.views import index, UserDeleteView, UserUpdateView, UserCreateView, UserDetails
urlpatterns = [
    path(r'', index),

    #Mascotas
    path('mascotas/', MascotasListView.as_view(), namespace='mascotas-list'),
    path('mascotas/<int:pk>/', MascotasDetailView.as_view(), namespace='mascotas-detail'),
    path('mascotas/add/', MascotasCreateView.as_view(), namespace='mascotas-add'),
    path('mascotas/<int:pk>/edit/', MascotasUpdateView.as_view(), namespace='mascotas-update'),
    path('mascotas/<int:pk>/delete/', MascotasDeleteView.as_view(), namespace='mascotas-dele
```

Nos dirigimos a views.py e importamos.

```
from django.views.generic import ListView, DetailView
from django.views.generic.edit import CreateView, DeleteView, UpdateView
from ilmp_app.models import User, Mascotas, Encuentros, Perdidos
from django.urls import reverse_lazy
```

Añadimos las vistas de mascotas, encuentros y perdidos en views.

```
#Mascotas

class MascotasListView(ListView):
    model = Mascotas

class MascotasDetailView(DetailView):
    model = Mascotas

class MascotasCreateView(CreateView):
    model = Mascotas
    fields = ['namePet', 'infoPet', 'agePet', 'typePet', 'imgPet', 'genderPet', 'usrPet']
    success_url = reverse_lazy('mascotas-list')

class MascotasUpdateView(UpdateView):
    model = Mascotas
    fields = ['namePet', 'infoPet', 'agePet', 'typePet', 'imgPet', 'genderPet', 'usrPet']
    template_name_sufix = '_update_form'

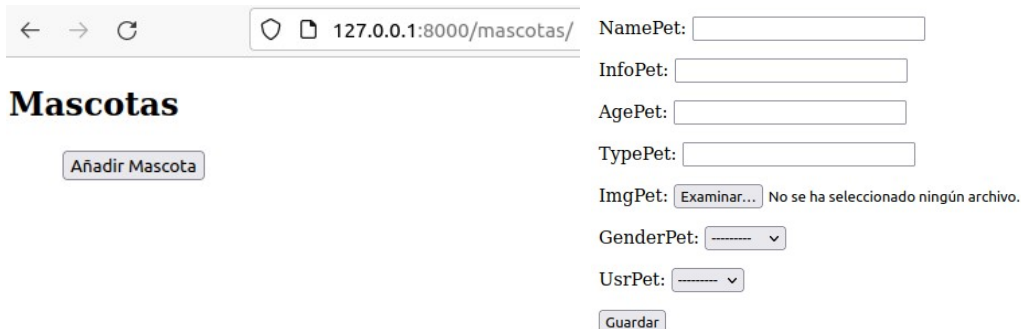
class MascotasDeleteView(DeleteView):
    model = Mascotas
    success_url = reverse_lazy('mascotas-list')
```

(agregar success\_url a update)

Añadimos a admin.py

```
2 from .models import *
3
4 admin.site.register(Mascotas)
5 admin.site.register(Perdidos)
6 admin.site.register(Encuentros)
7 admin.site.register(User)
```

Obtendremos un resultado como:



← → ↻ 127.0.0.1:8000/mascotas/

## Mascotas

[Añadir Mascota](#)

NamePet:

InfoPet:

AgePet:

TypePet:

ImgPet:  No se ha seleccionado ningún archivo.

GenderPet:

UsrPet:

Realizamos mismo proceso para Perdidos Y Encuentros y modificamos el index para poder acceder a ellos mediante hipervínculos.

Ahora realizaremos la extensión de usuario, para ello importaremos en admin.py

```
from django.contrib.auth.models import User
from django.contrib.auth.admin import UserAdmin as BaseUserAdmin
```

Importaremos AbstractUser y modificaremos Users de models.py

```
1  from django.db import models
2  from django.contrib.auth.models import AbstractUser
3
4
5  GENDER_CHOICES = (
6      ('M', 'Male'),
7      ('F', 'Female'),
8  )
9
10 #Usuarios
11 class User(AbstractUser):
12
13     name_usr = models.CharField(max_length=200)
14     gender_usr = models.CharField(max_length=1, choices=GENDER_CHOICES, blank=True)
15     birth_usr = models.DateField(null=True)
16     tel_usr = models.CharField(max_length = 9)
17     img_usr = models.ImageField(null=True, blank=True, upload_to="images/")
18     ubi_usr = models.CharField(max_length=200)
19
20     def __str__(self):
21         return self.name_usr
```

Habiendo hecho la extensión de usuario realizaremos el login/logout y register.

Importamos en urls views y añadimos

```
#Accounts
path("accounts/", include("django.contrib.auth.urls")),
path("register/", views.register_request, name="register"),
```

Creamos register\_request en el views que importamos.

```
#RegisterRequest
def register_request(request):
    if request.method == "POST":
        form = NewUserForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)
            messages.success(request, "Registro completado." )
            return redirect("main:home")
        messages.error(request, "Fallo en el registro, informacion invalida.")
    form = NewUserForm()
    return render (request=request, template_name="/register.html", context={"register_form":form})
```



Importamos en views.

```
from .forms import NewUserForm
from django.shortcuts import render, redirect
from django.contrib.auth import login, authenticate
from django.contrib import messages
from .forms import UserForm
from django.contrib.auth.mixins import LoginRequiredMixin, UserPassesTestMixin
```

Creamos forms.py en ilmp\_app y añadimos lo siguiente.

```
ilmp_app > forms.py > ...
1 from django import forms
2 from django import forms
3 from django.contrib.auth.forms import UserCreationForm
4
5
6 class ContactForm(forms.Form):
7     name = forms.CharField()
8     message = forms.CharField(widget=forms.Textarea)
9
10    def send_email(self):
11        # send email using the self.cleaned_data dictionary
12        pass
13
14    class UserForm(UserCreationForm):
15        class Meta:
16            model = User
17            fields = ('username', 'password1', 'password2', 'genderUsr', 'birthUsr', 'telUsr', 'img
18
19
20    class NewUserForm(UserCreationForm):
21        email = forms.EmailField(required=True)
22
23        class Meta:
24            model = User
25            fields = ("username", "email", "password1", "password2")
26
27        def save(self, commit=True):
28            user = super(NewUserForm, self).save(commit=False)
29            user.email = self.cleaned_data['email']
30            if commit:
31                user.save()
32            return user
```

Añadimos en setting.py

```
AUTH_USER_MODEL = 'ilmp_app.User'
LOGIN_REDIRECT_URL = "/"
LOGOUT_REDIRECT_URL = "/"
```

Por último creamos los ficheros login y register dentro de una carpeta llamada registration y añadimos lo siguiente.

```
ilmp_app > templates > registration > login.html > h2
1 <h2>Log In</h2>
2 <form method="post">
3     {% csrf_token %}
4     {{ form.as_p }}
5     <button type="submit">Log In</button>
6 </form>

ilmp_app > templates > registration > register.html > ...
1 {% block content %}
2
3 <!--Register-->
4 <div class="container py-5">
5     <h1>Register</h1>
6     <form method="POST">
7         {% csrf_token %}
8         {{ register_form|crispy }}
9         <button class="btn btn-primary" type="submit">Register</button>
10    </form>
11    <p class="text-center">If you already have an account, <a href="/login">login</a> instead.</p>
12 </div>
13
14 {% endblock %}
```

Modificamos un poco el Index para visualizar el Login/Logout y Register.

```
ilmp_app > templates > index.html > div
1 {% extends 'base.html' %}
2 {% load static %}
3
4 {% block content %}
5
6     {% if user.is_authenticated %}
7     <div>
8         Logeado como {{user.username}}
9         <a href="{% url 'logout' %}">Desloguear</a>
10    </div>
11    {% else %}
12    <div>
13        <a href="{% url 'login' %}">Logear</a>
14    </div>
15    <div>
16        <a moz-do-not-send="true" href="register/">Registrarse</a>
17    </div>
18    {% endif %}
19
20    Página principal en construcción
21
22    <br>
23    Apartados:
24    <br>
25    <a moz-do-not-send="true" href="mascotas">Mascotas</a><br>
26    <a moz-do-not-send="true" href="encuentros">Encuentros</a><br>
27    <a moz-do-not-send="true" href="perdidos">Perdidos</a><br>
28
29    {% endblock %}
```

Hecho esto visualizaremos la aplicación de este modo.



## Log In

Username:

Password:

Arreglamos para el registro en views:

```
class UserCreateView(CreateView):
    model = User
    form_class = UserForm
    success_url = reverse_lazy(index)
```

Añadimos en urls de ilmp y un formulario user\_forms.html.

```
path("register/", UserCreateView.as_view(), name='user-add'),
```

```
ilmp_app > templates > ilmp_app > <> user_form.html > form
1
2 <form method="post">{% csrf_token %}
3     {{ form.as_p }}
4     <input type="submit" value="Registrarse">
5 </form>
```

Y visualizamos que funciona.

Username:  Required. 15

Password:


- Your password can't be too similar to your oth

Para visualizar el nombre de usuario importamos en models y añadimos en el apartado usuarios:

```
from django.urls import reverse

21 def get_absolute_url(self):
22     return reverse('user-list', args=[self.pk])
```

Para visualizar las imagenes haremos uso de los ficheros estáticos y media.

Imagen : 

Para ello nos dirigimos a settings.py e importamos y añadimos lo siguiente.

```
14 import os

121 STATIC_URL = 'static/'
122 MEDIA_URL = '/media/'
123
124 MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
125
126 STATICFILES_DIRS = (
127     os.path.join(BASE_DIR, 'static'),
128 )
```

En urls de ilmp\_app importamos y añadimos.

```
from django.conf import settings
from django.conf.urls.static import static

32
33 urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
34
35
```

Añadimos los directorios static y media

```
> media
> static
```

Refrescamos la vista de la mascota y se muestra con su imagen.



Imagen :



Para que el usuario pueda visualizar, añadir, modificar y eliminar sus mascotas deberá estar logeado primero, para ello importaremos y modificaremos lo siguiente en views.

```
from django.contrib.auth.mixins import LoginRequiredMixin
```

```
class MascotasListView(LoginRequiredMixin,ListView):
```

En las vistas que queramos que se requiera de Login

Para lograr que, al registrar una mascota con tu nombre de usuario modificaremos nuestra view

```
#Mascotas
def creamascota(request):
    if request.method=="POST":
        var_mascota = MascotasForm(data=request.POST)
        if var_mascota.is_valid():
            print(request.user)
            usuario=User.objects.filter(pk=request.user.id)
            #print(usuario.username)
            mascota=var_mascota.save(commit=False)
            mascota.user=request.user
            mascota.save()
            return redirect('mascotas-list')
        else:
            var_mascota=MascotasForm()
            return render(request,"ilmp_app/mascotas_form.html",{"var_mascota":var_mascota})

class MascotasListView(LoginRequiredMixin,ListView):
    model = Mascotas

class MascotasDetailView(LoginRequiredMixin,DetailView):
    model = Mascotas

#class MascotasCreateView(LoginRequiredMixin,CreateView):
#    model = Mascotas
#    fields = ['namePet', 'infoPet', 'agePet', 'typePet', 'imgPet', 'genderPet']
#    success_url = reverse_lazy('mascotas-list')
```

Importaremos los recursos necesarios para realizar dicha función y modificaremos nuestro urls y form.

```
path("mascotas/add/", creamascota, name='mascotas-add'),
#path("mascotas/add/", MascotasCreateView.as_view(), name='mascotas-add'),
```

```
class MascotasForm(forms.ModelForm):
    class Meta:
        model = Mascotas
        fields = ['namePet', 'infoPet', 'agePet', 'typePet', 'imgPet', 'genderPet']
```

Hecho esto nos aparecerá el nombre del usuario que cree la mascota al guardarse.

Ahora queremos evitar que otro usuario pueda visualizar el contenido o las mascotas de otro y que solo pueda visualizar las suyas.

Para visualizar solo sus mascotas modificamos views y url de ilmp\_app e importamos.

```
def listamascota(request):
    listamascota=Mascotas.objects.filter(usrPet=request.user)
    return render(request,"ilmp_app/mascotas_list.html",{"listamascota":listamascota})

#class MascotasListView(LoginRequiredMixin,ListView):
#    model = Mascotas

#Mascotas
path('mascotas/', listamascota, name='mascotas-list'),
#path('mascotas/', MascotasListView.as_view(), name='mascotas-list'),
```

De este modo solo visualizará sus mascotas, aún así podrá acceder mediante la url a la información de otras por lo que añadimos el siguiente fichero llamado decorators.py con el siguiente contenido.

```
ilmp_app > decorators.py > check_pet_owner
1 from django.http import HttpResponse
2 from ilmp_app.models import Mascotas
3
4 def check_pet_owner(func):
5     def check_and_call(request, *args, **kwargs):
6         pk = kwargs["pk"]
7         mascotas = Mascotas.objects.get(pk=pk)
8         if not (mascotas.usrPet.id == request.user.id):
9             return HttpResponse("No puedes acceder a este sitio, no tienes permiso.", status=403)
10        return func(request, *args, **kwargs)
11    return check_and_call
```

Sobre las views que no queramos que otro usuario que no sea el mismo propietario no tenga acceso situaremos lo siguiente e importamos los recursos necesarios.

```
@method_decorator(check_pet_owner,name='dispatch')
class MascotasDetailView(LoginRequiredMixin,DetailView):
    model = Mascotas

#class MascotasCreateView(LoginRequiredMixin,CreateView):
#    model = Mascotas
#    fields = ['namePet', 'infoPet', 'agePet', 'typePet', 'imgPet', 'genderPet']
#    success_url = reverse_lazy('mascotas-list')

@method_decorator(check_pet_owner,name='dispatch')
class MascotasUpdateView(LoginRequiredMixin,UpdateView):
    model = Mascotas
    fields = ['namePet', 'infoPet', 'agePet', 'typePet', 'imgPet', 'genderPet']
    template_name_sufix = '_update_form'
    success_url = reverse_lazy('mascotas-list')

@method_decorator(check_pet_owner,name='dispatch')
class MascotasDeleteView(LoginRequiredMixin>DeleteView):
    model = Mascotas
    success_url = reverse_lazy('mascotas-list')
```

De este modo el usuario no puede acceder a una mascota que no sea propia.



No puedes acceder a este sitio, no tienes permiso.

Para que en una pérdida aparezca una mascota propia haremos lo siguiente.

Falta continuar con este apartado

Para añadir una plantilla de bootstrap buscamos alguna gratuita y la acoplamos a nuestro proyecto.

Una vez modificado el estilo y el index añadimos en views el buscador e importamos Q.

```
#Buscador
class search(ListView):
    model = Perdidos
    template_name="search.html"
    def get_queryset(self):
        query = self.request.GET.get("q")
        object_list=Perdidos.objects.filter(Q(petLost__in=[query]))
        return object_list
```

Creamos un formulario nuevo que nos muestre las coincidencias del search

```
{% extends 'base.html' %}
{% block content %}

{% for contenido in object_list %}
    <a href={% url 'perdidos-detail' contenido.id %}>{{ contenido.petLost }}</a>
{%endfor%}

{% endblock %}
```

Este actualmente filtra por id de la mascota perdida en proceso de cambio.

Arreglamos un error en views de mascotas en creamascota que no nos permitía guardar las imagenes añadiendo request.Files.

```
#Mascotas
def creamascota(request):
    if request.method=="POST":
        var_mascota = MascotasForm(request.POST, request.FILES)
```

Añadimos enctype a nuestro formulario de usuario que hacía que ocurriese lo mismo.

```
<form enctype="multipart/form-data"
```



Para el uso de la API nos dirigimos a settings.py y añadimos rest\_framework en installed\_apps y el contenido de REST\_FRAMEWORK al final.

```
INSTALLED_APPS = [
    'ilmp_app.apps.ILmpAppConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
]
```

```
#Api
REST_FRAMEWORK = {
    # Use Django's standard `django.contrib.auth` permissions,
    # or allow read-only access for unauthenticated users.
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.DjangoModelPermissionsOrAnonReadOnly'
    ]
}
```

En ilmp\_app creamos un fichero api.py en el que añadimos el siguiente contenido.

```
ilmp_app > api.py > EncuentrosSerializer > Meta
1  ##Api
2
3  from django.contrib.auth.models import User
4  from .models import *
5  from rest_framework import routers, serializers, viewsets
6
7  ###Usuarios
8
9  # Serializers define the API representation.
10 class UserSerializer(serializers.HyperlinkedModelSerializer):
11     class Meta:
12         model = User
13         fields = ['url', 'username', 'email', 'is_staff']
14
15 # ViewSets define the view behavior.
16 class UserViewSet(viewsets.ModelViewSet):
17     queryset = User.objects.all()
18     serializer_class = UserSerializer
19
20 ###Mascotas
21
22 # Serializers define the API representation.
23 class MascotasSerializer(serializers.HyperlinkedModelSerializer):
24     class Meta:
25         model = Mascotas
26         fields = ['url', 'namePet', 'infoPet', 'agePet', 'typePet', 'imgPet', 'genderPet', 'u
27
28 # Serializers define the API representation.
29 class PerdidosSerializer(serializers.HyperlinkedModelSerializer):
30     class Meta:
31         model = Perdidos
32         fields = ['infoLost', 'dateLost', 'petLost', 'ubiLost']
33
34 # ViewSets define the view behavior.
35 class PerdidosViewSet(viewsets.ModelViewSet):
36     queryset = Perdidos.objects.all()
37     serializer_class = PerdidosSerializer
38
39 ###Encuentros
40
41 # Serializers define the API representation.
42 class EncuentrosSerializer(serializers.HyperlinkedModelSerializer):
43     class Meta:
44         model = Encuentros
45         fields = ['typeFind', 'imgFind', 'infoFind', 'genderFind', 'ubiFind']
46
47 # ViewSets define the view behavior.
48 class EncuentrosViewSet(viewsets.ModelViewSet):
49     queryset = Encuentros.objects.all()
50     serializer_class = EncuentrosSerializer
51
52 #Api Reg
53 router = routers.DefaultRouter()
54 router.register(r'users', UserViewSet)
55 router.register(r'mascotas', MascotasViewSet)
56 router.register(r'perdidos', PerdidosViewSet)
57 router.register(r'encuentros', EncuentrosViewSet)
```

En la url importamos

```
from ilmp_app.api import router
```

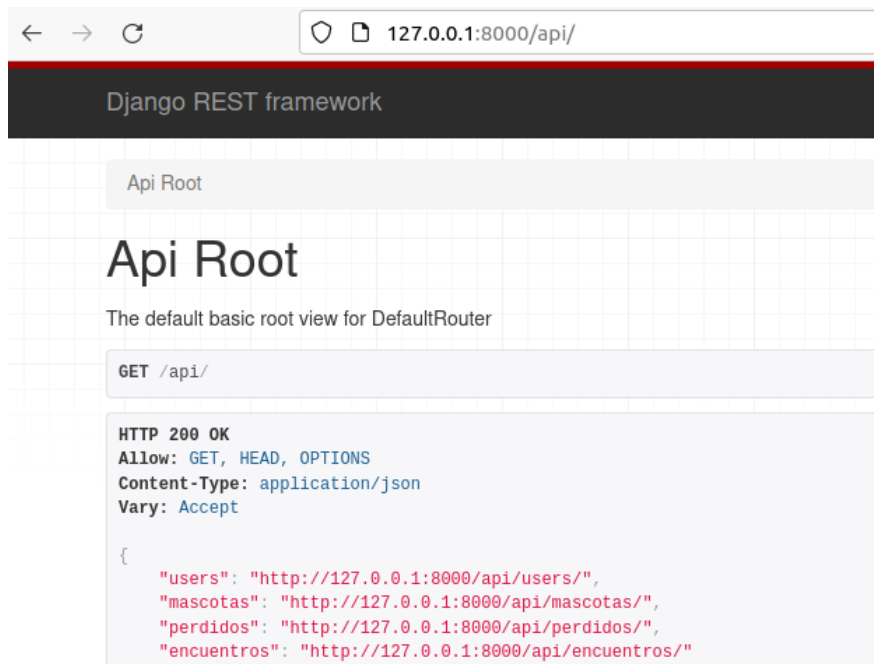
Y añadimos la ruta.

```
#Api
path('api/', include(router.urls)),
path('api-auth/', include('rest_framework.urls', namespace='rest_framework')),
```

Si no se tiene instalado, instalar el siguiente módulo.

```
^C(ilmp-env) sacopapa@sacopapa-HP-Laptop-15-da0xxx:~/Escritorio1/Escritorio/AreaTrabajo/Nuevo_Area/AplicacionWeb/ilmp$ pip3 install django
framework
```

Hecho esto debería mostrarse algo como esto.



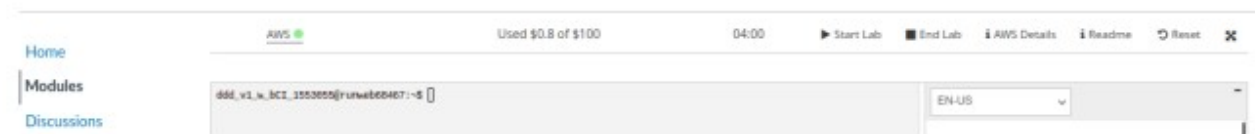
Para que los formularios no nos redirijan a la api de cambiaremos la url

```
#path(r'', include('ilmp_app.urls')),
path('', include(('ilmp_app.urls','ilmp'),namespace="ilmp")),
```

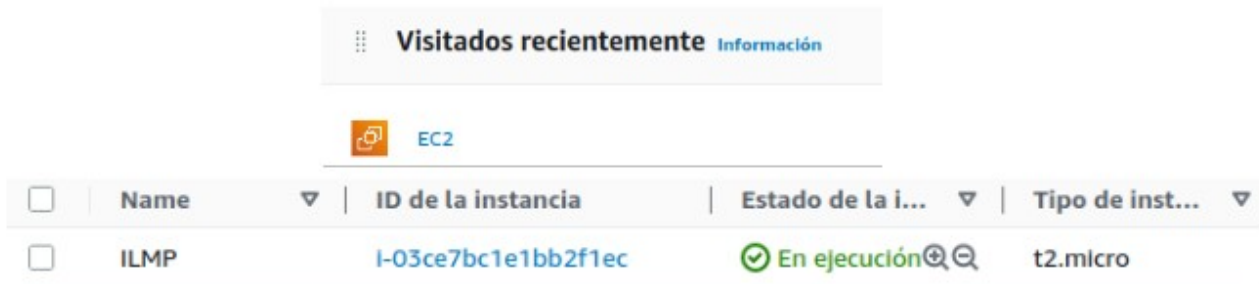
y en cada redirección añadiremos ilmp como se muestra a continuación.

```
<li><a href={% url 'ilmp:encuentros-detail' %}>
  <a href="{% url 'ilmp:encuentros-update' %}">
  <a href="{% url 'ilmp:encuentros-delete' %}">
template_name_sufix = '_update_form'
success_url = reverse_lazy('ilmp:mascotas-l
```

Levantamos nuestra máquina en AWS



Accedemos a la instancia creada EC2



Al crear la instancia generaremos el par de claves para acceder mediante ssh a la instancia, de no haberlo creado junto la instancia nos dirigiremos a Red y seguridad/Par de claves y generamos.

☐ keys rsa 2022/11/11 16:34 GMT+1

Descargamos las claves y modificaremos los permisos con chmod para hacer uso de ellas.

```
-r----- 1 sacopapa sacopapa 1674 nov 11 16:34 keys.pem
```

En el resumen de nuestra instancia buscaremos el DNS de IPv4 pública y copiaremos en el portapapeles para poder conectarnos a la máquina con nuestra clave a dicha dirección.

DNS de IPv4 pública

ec2-34-205-25-136.compute-1.amazonaws.com | [dirección abierta](#)

```
sacopapa@sacopapa-HP-Laptop-15-da0xxx:~/Descargas$ ssh -i "keys.pem" ubuntu@ec2-34-205-25-136.compute-1.amazonaws.com
The authenticity of host 'ec2-34-205-25-136.compute-1.amazonaws.com (34.205.25.136)' can't be established.
ECDSA key fingerprint is SHA256:JLUFHY/QN2JMSHPB+t9KF3GIZRww+8vBzpBf84lLoRc.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

Nos dirigimos al directorio /var/www y colocaremos en esa ubicación nuestra aplicación.

```
ubuntu@ip-172-31-91-31:/var/www$ sudo git clone https://github.com/LmGra/IAW_ILostMyPet
```

Instalamos el entorno virtual, creamos y lo usamos

```
ubuntu@ip-172-31-91-31:/var/www$ sudo apt install python3-virtualenv
ubuntu@ip-172-31-91-31:/var/www$ sudo virtualenv ven
ubuntu@ip-172-31-91-31:/var/www$ source ven/bin/activate
(ven) ubuntu@ip-172-31-91-31:/var/www$
```



Instalamos en el entorno virtual todos los módulos que usaremos y nginx.

```
(ven) ubuntu@ip-172-31-91-31:/var/www$ sudo pip3 install django-rest-framework
(ven) ubuntu@ip-172-31-91-31:/var/www$ sudo pip3 install django-crispy-forms
(ven) ubuntu@ip-172-31-91-31:/var/www$ sudo pip3 install django
(ven) ubuntu@ip-172-31-91-31:/var/www$ sudo pip3 install django-extensions
(ven) ubuntu@ip-172-31-91-31:/var/www$ sudo apt install nginx
```

Configuramos el nginx para nuestro sitio, para ello deberemos crear un fichero llamado uwsgi\_params y añadir el siguiente contenido en el directorio del proyecto.

```
(ven) ubuntu@ip-172-31-91-31:/var/www$ sudo nano IAW_ILostMyPet/ILMP/uwsgi_params
uwsgi_param QUERY_STRING      $query_string;
uwsgi_param REQUEST_METHOD    $request_method;
uwsgi_param CONTENT_TYPE      $content_type;
uwsgi_param CONTENT_LENGTH    $content_length;

uwsgi_param REQUEST_URI       $request_uri;
uwsgi_param PATH_INFO          $document_uri;
uwsgi_param DOCUMENT_ROOT     $document_root;
uwsgi_param SERVER_PROTOCOL    $server_protocol;
uwsgi_param REQUEST_SCHEME    $scheme;
uwsgi_param HTTPS              $https if_not_empty;

uwsgi_param REMOTE_ADDR        $remote_addr;
uwsgi_param REMOTE_PORT        $remote_port;
uwsgi_param SERVER_PORT        $server_port;
uwsgi_param SERVER_NAME        $server_name;
```

Creamos un fichero llamado ilmp\_nginx.conf en el directorio /etc/nginx/sites-available/ y añadimos el siguiente contenido.

```
GNU nano 6.2 /etc/nginx/sites-available/ilmp_nginx.conf
# the upstream component nginx needs to connect to
upstream django {
    # server unix:///path/to/your/mysite/mysite.sock; # for a file socket
    server 127.0.0.1:8001; # for a web port socket (we'll use this first)
}

# configuration of the server
server {
    # the port your site will be served on
    listen 8000;
    # the domain name it will serve for
    server_name ilmp_app.duckdns.org; # substitute your machine's IP address or FQDN
    charset utf-8;

    # max upload size
    client_max_body_size 75M; # adjust to taste

    # Django media
    location /media {
        alias /www/var/IAW_ILostMyPet/ILMP/media; # your Django project's media files - amend as required
    }

    location /static {
        alias /www/var/IAW_ILostMyPet/ILMP/static; # your Django project's static files - amend as required
    }

    # Finally, send all non-media requests to the Django server.
    location / {
        uwsgi_pass django;
        include /www/var/IAW_ILostMyPet/ILMP/uwsgi_params; # the uwsgi_params file you installed
    }
}
```

(Comentamos listen 8000 y arreglo server\_name ilmpapp)



Reiniciamos el servicio de nginx y generamos un enlace simbólico.

```
(ven) ubuntu@ip-172-31-91-31:/var/www$ sudo /etc/init.d/nginx restart
Restarting nginx (via systemctl): nginx.service.
(ven) ubuntu@ip-172-31-91-31:/var/www$ sudo ln -s /etc/nginx/sites-available/ilmp.nginx.conf /etc/nginx/sites-enabled/
```

Modificamos el setting de nuestra aplicación y añadimos:

```
STATIC_ROOT = os.path.join(BASE_DIR, "static/")
```

Hecho esto realizamos un collectstatic, ya que hay que hacerlo manualmente porque no haremos uso de runserver.

```
(ven) ubuntu@ip-172-31-91-31:/var/www$ python3 IAW_ILostMyPet/ILMP/manage.py collectstatic

You have requested to collect static files at the destination
location as specified in your settings:

    /var/www/IAW_ILostMyPet/ILMP/static

This will overwrite existing files!
Are you sure you want to do this?

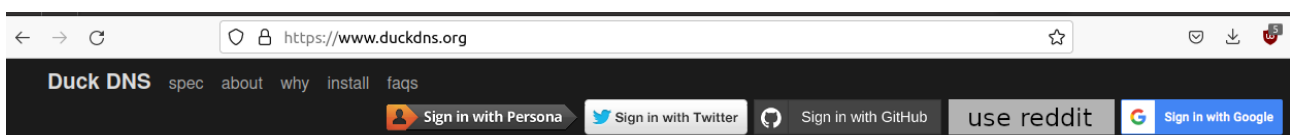
Type 'yes' to continue, or 'no' to cancel: yes

165 static files copied to '/var/www/IAW_ILostMyPet/ILMP/static'.
```

Configuramos las reglas de entrada en nuestra instancia de AWS

| ID de la regla del grupo d... | Intervalo de pu... | Protocolo | Orig  |
|-------------------------------|--------------------|-----------|-------|
| sgr-009734de717abcfbc         | Todo               | ICMP      | 0.0.0 |
| sgr-0843f5e7d8ba3ae6d         | 443                | TCP       | 0.0.0 |
| sgr-0990590ca2d95cf6c         | 8000               | TCP       | 0.0.0 |
| sgr-05461d8667fabbd17         | 80                 | TCP       | 0.0.0 |
| sgr-0e16f78a67332c901         | 8080               | TCP       | 0.0.0 |
| sgr-0b146e916889470f7         | 22                 | TCP       | 0.0.0 |

Para hostear en aws con dns gratuito haremos uso de duckdns, accedemos mediante gmail.



Añadimos un dominio.

| domain  | current ip                              | ipv6  | changed                                   |
|---------|---|---|---|
| ilmpapp | 54.172.66.191 <a href="#">update ip</a> | <input type="text" value="ipv6 address"/> <a href="#">update ipv6</a> | 3 weeks ago <a href="#">delete domain</a> |

Vamos al apartado instalación, seleccionamos nuestro dominio y seguimos los pasos que aparecen en la página.

Para la instalación y uso de certbot.

```
^Cubuntu@ip-172-31-91-31:~/ilmp/ilmp$ sudo apt install snapd
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
snapd is already the newest version (2.57.5+22.04ubuntu0.1).
snapd set to manually installed.
The following packages were automatically installed and are no longer required:
  libflashrom1 libftdi1-2
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 14 not upgraded.
ubuntu@ip-172-31-91-31:~/ilmp/ilmp$ sudo snap install --classic certbot
certbot 1.32.1 from Certbot Project (certbot-eff✓) installed
ubuntu@ip-172-31-91-31:~/ilmp/ilmp$ sudo ln -s /snap/bin/certbot /usr/bin/certbot
```

Una vez instalado haremos uso del comando certbot --nginx y añadiremos nuestro correo y seleccionaremos nuestro dominio, estas líneas se añadirá automáticamente a nuestro fichero `etc/nginx/sites-available/default`

```
listen 443 ssl; # managed by Certbot
ssl_certificate /etc/letsencrypt/live/ilmpapp.duckdns.org/fullchain.pem; # >
ssl_certificate_key /etc/letsencrypt/live/ilmpapp.duckdns.org/privkey.pem; # >
include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

}

server {
    if ($host = ilmpapp.duckdns.org) {
        return 301 https://$host$request_uri;
    } # managed by Certbot
}
```

Para levantar nuestra aplicación con nginx y wsgi haremos lo siguiente:

```
ubuntu@ip-172-31-91-31:/var/www$ sudo su -l www-data -s /bin/bash
www-data@ip-172-31-91-31:~$ ls
basura  html  ilmp  mysite  papelera  static_env  ven
www-data@ip-172-31-91-31:~$ source ven/bin/activate
(ven) www-data@ip-172-31-91-31:~$ cd ilmp
(ven) www-data@ip-172-31-91-31:~/ilmp$ uwsgi --socket 127.0.0.1:8001 --module ilmp.wsgi
*** Starting uWSGI 2.0.21 (64bit) on [Fri Jan 13 14:00:02 2023] ***
```

Debemos tener en cuenta que los permisos están asignados a www-data.

Ahora accederemos con el navegador a nuestra aplicación escribiendo la dirección de nuestro dns.



Para utilizar postgres como gestor de base de datos de nuestra aplicación lo instalamos y levantamos el servicio.

```
(ilmp-env) sacopapa@sacopapa-HP-Laptop-15-da0xxx:~/Escritorio1/Escritorio/AreaTrabajo/Nuevo_Area/AplicacionWeb/ilmp$ sudo apt install postgresql postgresql-contrib
```

Para empezar a usar Postgres lo lanzamos con: **sudo -u postgres psql** creamos una database ilmp y un usuario al que se la asignamos

```
(ven) ubuntu@ip-172-31-91-31:/var/www$ sudo -u postgres psql
psql (14.6 (Ubuntu 14.6-0ubuntu0.22.04.1))
Type "help" for help.

postgres=# CREATE DATABASE ilmp;
CREATE DATABASE
postgres=# CREATE USER ilmpuser WITH PASSWORD 'ilmpuser';
CREATE ROLE
postgres=# ALTER ROLE ilmpuser SET client_encoding TO 'utf8';
ALTER ROLE
postgres=# ALTER ROLE ilmpuser SET default_transaction_isolation TO 'read committed';
ALTER ROLE
postgres=# ALTER ROLE ilmpuser SET timezone TO 'UTC';
ALTER ROLE
postgres=# GRANT ALL PRIVILEGES ON DATABASE ilmp TO ilmpuser;
GRANT
postgres=# exit
```

También deberemos instalar el plugin para conectar nuestra aplicación con la base, para ello instalamos el paquete psycopg2-binary con pip3.

Hecho esto toca modificar nuestro fichero setting.py.

```
DATABASES = {
    'default': {
        #'ENGINE': 'django.db.backends.sqlite3',
        #'NAME': BASE_DIR / 'db.sqlite3',
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'ilmp',
        'USER': 'ilmpuser',
        'PASSWORD': 'ilmpuser',
        'HOST': 'localhost',
        'PORT': '',
    }
}
```



Si todo sale bien podemos visualizar las tablas de nuestra aplicación y su contenido.

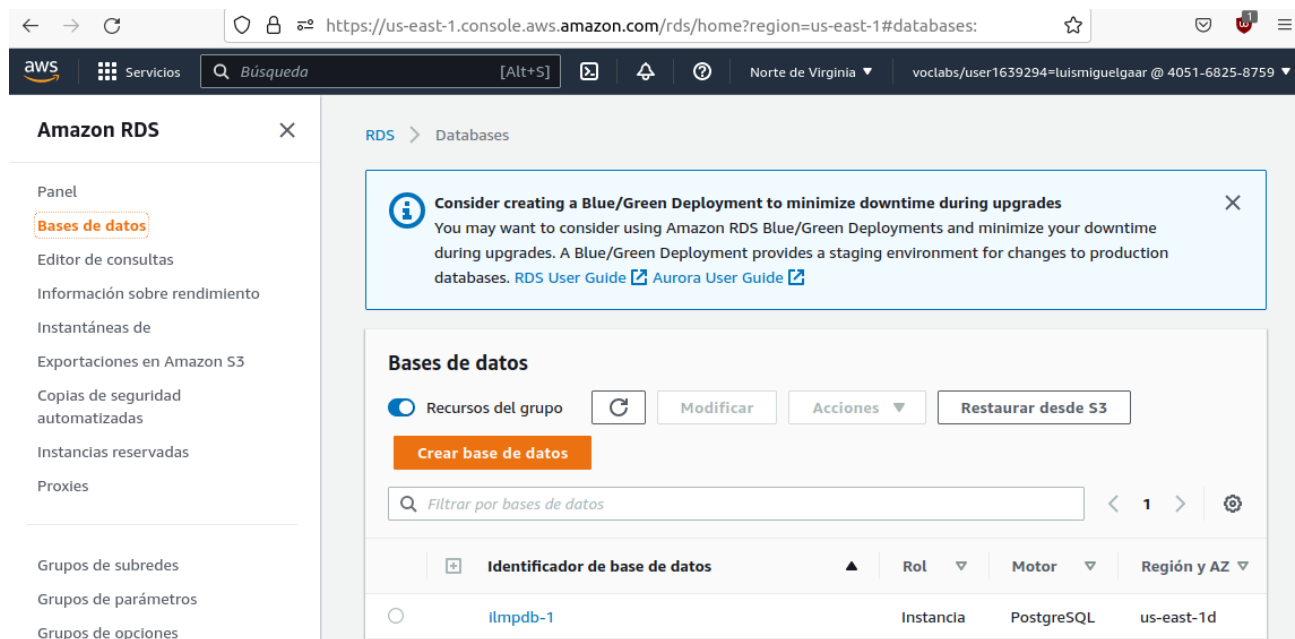
```
(ven) ubuntu@ip-172-31-91-31:/var/www$ sudo -u postgres psql
psql (14.6 (Ubuntu 14.6-0ubuntu0.22.04.1))
Type "help" for help.

postgres=# \connect ilmp
now connected to database "ilmp" as user "postgres".
Terminal \l
                                List of databases
  Name      | Owner   | Encoding | Collate | Ctype   | Access privileges
-----+-----+-----+-----+-----+-----
  ilmp       | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =Tc/postgres +
            |          |          |          |          | postgres=CTc/postgres+
            |          |          |          |          | ilmpuser=CTc/postgres
 postgres   | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres +
 template0  | postgres | UTF8     | C.UTF-8 | C.UTF-8 | postgres=CTc/postgres
 template1  | postgres | UTF8     | C.UTF-8 | C.UTF-8 | =c/postgres +
            |          |          |          |          | postgres=CTc/postgres
(4 rows)
```

```
ilmp=# \dt
                                List of relations
 Schema | Name | Type | Owner
-----+-----+-----+-----
 public | auth_group | table | ilmpuser
 public | auth_group_permissions | table | ilmpuser
 public | auth_permission | table | ilmpuser
 public | django_admin_log | table | ilmpuser
 public | django_content_type | table | ilmpuser
 public | django_migrations | table | ilmpuser
 public | django_session | table | ilmpuser
 public | ilmp_app_correo | table | ilmpuser
 public | ilmp_app_encuentros | table | ilmpuser
 public | ilmp_app_mascotas | table | ilmpuser
 public | ilmp_app_perdidos | table | ilmpuser
 public | ilmp_app_user | table | ilmpuser
 public | ilmp_app_user_groups | table | ilmpuser
 public | ilmp_app_user_user_permissions | table | ilmpuser
(14 rows)
```

```
ilmp=# select * from ilmp_app_perdidos;
 id | infoLost | dateLost | ubiLost
-----+-----+-----+-----
  1 | Se me ha perdido el perro, es verde con puntos rosas | 2023-01-12 00:00:00+00 | La estación espacial de marte |
  1
(1 row)
```

Abrimos AWS y creamos una base de datos ilmpdb-1 postgres en RDS.



La vinculamos a nuestra EC2 y generaremos VPC por defecto.

Visualizamos la descripción

| Conectividad y seguridad   |  |  |
|--|--|--|
| Punto de enlace y puerto   | Redes  | Seguridad  |
| Punto de enlace<br>ilmpdb-1.ci5gitzmjdfr.us-east-1.rds.amazonaws.com | Zona de disponibilidad<br>us-east-1d               | Grupos de seguridad<br>la VPC<br>default<br>(sg-014c779a0f6) |
| Puerto<br>5432   | VPC<br>vpc-0583cc659b2430e61                       | ✓ Activo<br>rds-ec2-1<br>(sg-0485af49dfcc)                   |
|  | Grupo de subredes<br>default-vpc-0583cc659b2430e61 | ✓ Activo   |
|  |  | Accesible públicamente<br>No                                 |

Accedemos al RDS y creamos la base de datos y realizamos la misma operación que realizamos en local.

```
ubuntu@ip-172-31-91-31:/var/www$ sudo psql -U postgres -h ilmpdb-1.ci5gitzmjdfdr.us-east-1.rds.amazonaws.com
Password for user postgres:
psql (14.6 (Ubuntu 14.6-0ubuntu0.22.04.1), server 13.7)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.
```

Una vez creamos la base de datos modificaremos el setting.py de nuestra aplicación para conectar nuestra aplicación con el RDS.

```
DATABASES = {
    'default': {
        #'ENGINE': 'django.db.backends.sqlite3',
        #'NAME': BASE_DIR / 'db.sqlite3',
        # 'ENGINE': 'django.db.backends.postgresql_psycopg2',
        # 'NAME': 'ilmp',
        # 'USER': 'ilmpuser',
        # 'PASSWORD': 'ilmpuser',
        # 'HOST': 'localhost',
        # 'PORT': '',
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'ilmp',
        'USER': 'ilmpuser',
        'PASSWORD': 'ilmpuser',
        'HOST': 'ilmpdb-1.ci5gitzmjdfdr.us-east-1.rds.amazonaws.com',
        'PORT': '',
    }
}
```

Al acceder a nuestra base de datos veremos que está vacía, deberemos exportar la base de datos que tenemos en local para visualizar la información en el RDS.

Para ello volvemos a poner en el settings la ubicación en local de la bdd de postgres y haremos uso del comando que nos proporciona django: **python3 manage.py dumpdata > ilmp.json**

Hecho esto volvemos a modificar el setting.py para que se conecte al rds y para poblar nuestra bdd haremos uso del comando: **python3 manage.py loaddata ilmp.json**

Instalamos Wagtail y sus dependencias con pip y creamos un nuevo proyecto con wagtail.

Creamos página de inicio y de blog; templates/blog/blog\_index\_page.html

```
{% extends "base.html" %}

{% load wagtailcore_tags %}

{% block body_class %}template-blogindexpage{% endblock %}

{% block content %}
    <h1>{{ page.title }}</h1>

    <div class="intro">{{ page.intro|richtext }}</div>

    {% for post in page.get_children %}
        <h2><a href="{% pageurl post %}">{{ post.title }}</a></h2>
        {{ post.specific.intro }}
        {{ post.specific.body|richtext }}
    {% endfor %}
{% endblock %}
```

blog\_page.html

```
{% extends "base.html" %}

{% load wagtailcore_tags %}

{% block body_class %}template-blogpage{% endblock %}

{% block content %}
    <h1>{{ page.title }}</h1>
    <p class="meta">{{ page.date }}</p>

    <div class="intro">{{ page.intro }}</div>

    {{ page.body|richtext }}

    <p><a href="{% page.get_parent.url %}">Return to blog</a></p>
{% endblock %}
```

En apps.py de blog

```
1 from django.apps import AppConfig
2
3
4 class BlogAppConfig(AppConfig):
5     default_auto_field = 'django.db.models.BigAutoField'
6     name = 'blog'
```



Lo agregamos en installed apps de settings.

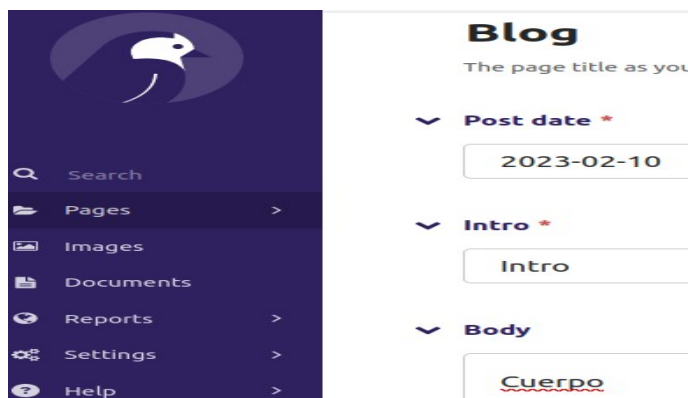
```
'blog.apps.BlogAppConfig',
```

Y añadimos el siguiente contenido en el modelo de blog

```
9  from django.db import models
10
11  from wagtail.models import Page
12  from wagtail.fields import RichTextField
13  from wagtail.admin.panels import FieldPanel
14  from wagtail.search import index
15
16  class BlogIndexPage(Page):
17      intro = RichTextField(blank=True)
18
19      content_panels = Page.content_panels + [
20          FieldPanel('intro')
21      ]
22
23  # Keep the definition of BlogIndexPage, and add:
24
25  class BlogPage(Page):
26      date = models.DateField("Post date")
27      intro = models.CharField(max_length=250)
28      body = RichTextField(blank=True)
29
30      search_fields = Page.search_fields + [
31          index.SearchField('intro'),
32          index.SearchField('body'),
33      ]
34
35      content_panels = Page.content_panels + [
36          FieldPanel('date'),
37          FieldPanel('intro'),
38          FieldPanel('body'),
39      ]
```


Añadimos las modificaciones a la url.

Abrimos nuestra aplicación y nos dirigimos al /cms, en page añadiremos child page-blog.



Ahora en el apartado de Wagtail/groups, añadimos un grupo Blogger que pueda añadir y editar.

### ➤ Page permissions

| Page  | Add                                 | Edit                                | Publish                  | Bulk delete              | Lock                     | Unlock                   |                        |
|---|-------------------------------------|-------------------------------------|--------------------------|--------------------------|--------------------------|--------------------------|------------------------|
|  Blog_Ilmp | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <a href="#">Delete</a> |

---

[+ Add a page permission](#)

Hecho blogger nos dirigimos a /admin y en el apartado groups visualizaremos los permisos.

Para acceder nos dirigimos a la dirección /blog y visualizamos.



Para conectar por LDAP instalamos ldap y sus dependencias, con docker bitnami/openldap

Realizamos mapeo de puertos.

```
ubuntu@ip-172-31-91-31:/var/www$ docker run -p 1389:1389 --detach --rm --name openldap --network my-network
--env LDAP_ADMIN_USERNAME=admin --env LDAP_ADMIN_PASSWORD=adminpassword --env LDAP_USERS=customuser
--env LDAP_PASSWORDS=custompassword bitnami/openldap:latest
```

Añadimos base de datos mariagalera.

```
ubuntu@ip-172-31-91-31:/var/www$ docker run --detach --rm --name mariadb-galera --network my-network
--env MARIADB_ROOT_PASSWORD=root-password --env MARIADB_GALERA_MARIABACKUP_PASSWORD=backup-password
--env MARIADB_USER=customuser --env MARIADB_DATABASE=customdatabase --env MARIADB_ENABLE_LDAP=yes
--env LDAP_URI=ldap://openldap:1389 --env LDAP_BASE=dc=example,dc=org --env LDAP_BIND_DN=cn=admin,dc=
example,dc=org --env LDAP_BIND_PASSWORD=adminpassword bitnami/mariadb-galera:latest
```

En nuestro setting.py añadimos

```
GNU nano 6.2      ilmp/ilmp/settings.py *
import ldap
from django_auth_ldap.config import LDAPSearch

# Baseline configuration.
AUTH_LDAP_SERVER_URI = "ldap://127.0.0.1:1389"

AUTH_LDAP_BIND_DN = "cn=admin,dc=example,dc=org"

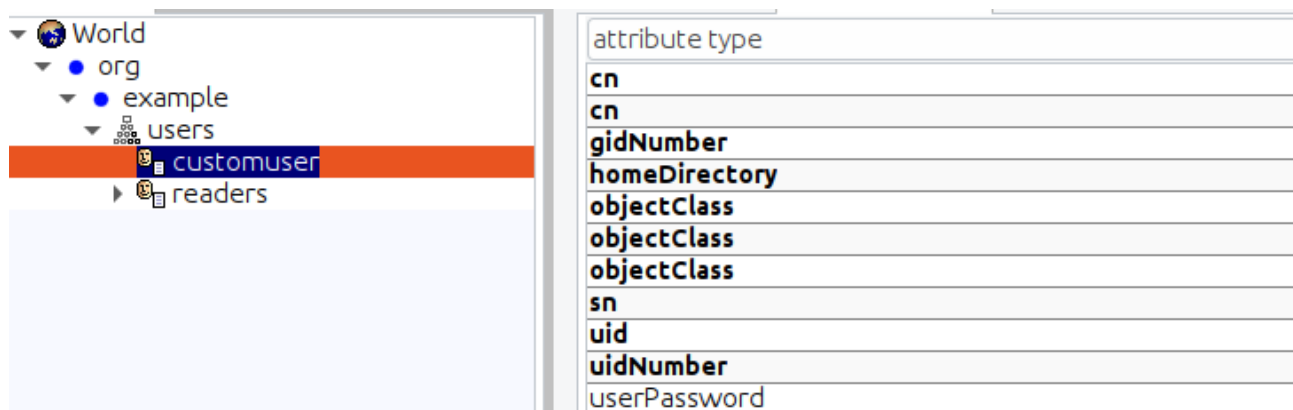
AUTH_LDAP_BIND_PASSWORD = "adminpassword"
AUTH_LDAP_USER_SEARCH = LDAPSearch(
    "ou=users,dc=example,dc=org", ldap.SCOPE_SUBTREE, "(uid=%(user)s)"
)

AUTH_LDAP_USER_ATTR_MAP = {
    "email": "mail",
}

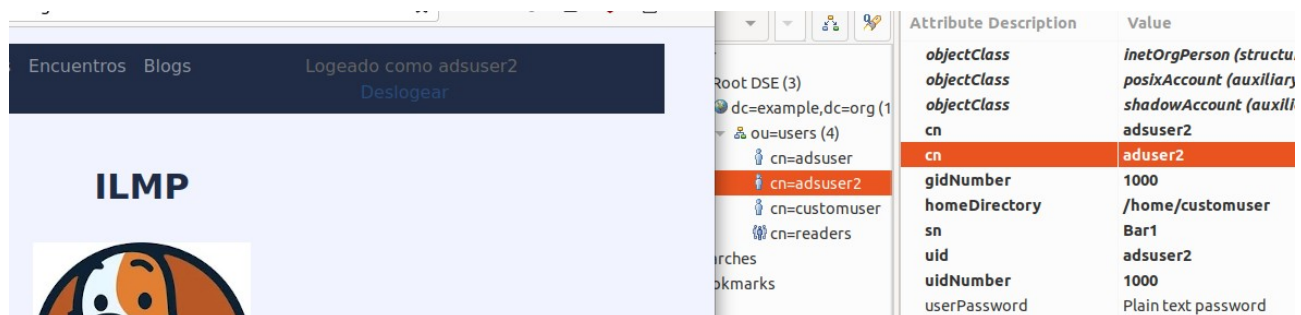
AUTHENTICATION_BACKENDS = (
    "django_auth_ldap.backend.LDAPBackend",
    "django.contrib.auth.backends.ModelBackend",
)
```

Nos dirigimos al jxplorer y visualizamos.

|                 |                            |       |      |
|-----------------|----------------------------|-------|------|
| Host:           | ilmpapp.duckdns.org        | Port: | 1389 |
| Protocol:       | LDAP v3                    |       |      |
| DSML Service:   |                            |       |      |
| Optional Values |                            |       |      |
| Base DN:        | cn=admin,dc=example,dc=org |       |      |
| Security        |                            |       |      |
| Level:          | User + Password            |       |      |
| User DN:        | cn=admin,dc=example,dc=org |       |      |
| Password:       | *****                      |       |      |



Modificamos o duplicamos customuser y modificamos nombre y contraseña para comprobar el usuario con ldap.



Ahora realizamos un script de automatización para agregar usuarios al grupo Blogger.

```
GNU nano 6.2 signals.py
from django.contrib.auth.models import Group
from django.core.management.base import BaseCommand
from django.contrib.auth.signals import user_logged_in
from ilmp_app.models import User

class Command(BaseCommand):

    def add_arguments(self, parser):
        parser.add_argument('usuario', type=str, help='Introduce el usuario a meter en Bloggers')

    def handle(self, *args, **kwargs):
        try:
            nombre = kwargs['usuario']
            u = User.objects.get(username=nombre)
            g = Group.objects.get(name='Blogger')
            if g in u.groups.all():
                print(f"{u.username} Ya pertenece al grupo.")
            else:
                print(f"{u.username} Añadiendo el usuario al grupo Blogger")
                u.groups.add(g)
        except:
            print("El usuario introducido no existe")
```

Para automatizarlo lo movememos a blog y le pondremos de nombre signals.py y añadimos def ready(self): import blog.signals en *blogs/apps.py*



Añadimos en el setting la ubicación en la que se nos va a almacenar el correo que se nos generará en forma de dichero.

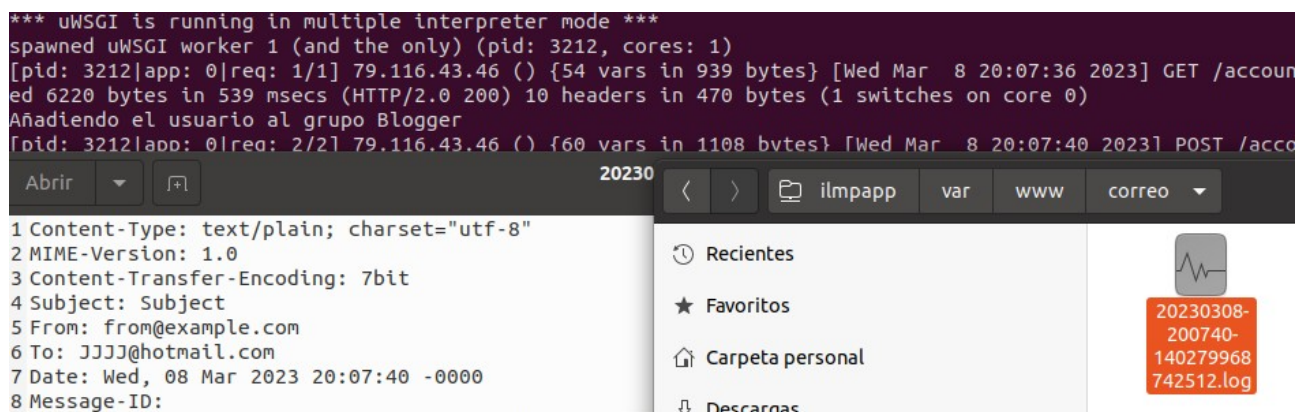
```
EMAIL_BACKEND = 'django.core.mail.backends.filebased.EmailBackend'
EMAIL_FILE_PATH = '/var/www/correo' # change this to a proper location
```

En Signals.py de Blog añadimos/modificamos lo siguiente para que al logear el usuario por primera vez sea añadido al grupo Blogger y se envíe el correo.

```
1 from django.contrib.auth.models import Group
2 from django.core.management.base import BaseCommand
3 from django.contrib.auth.signals import user_logged_in
4 from ilmp_app.models import User
5 from django.core.mail import send_mail
6 from django.dispatch import receiver
7
8 @receiver(user_logged_in)
9 def log_user_login(sender, user, **kwargs):
10
11     u=User.objects.get(username=user)
12     g=Group.objects.get(name='Blogger')
13     c=User.objects.get(username=user).email
14
15     if g in u.groups.all():
16         print("Este usuario ya pertenece al grupo Blogger")
17     else:
18         print("Añadiendo el usuario al grupo Blogger")
19         send_mail(
20             'Subject',
21             'Message',
22             'from@example.com',
23             [c],
24             fail_silently=False,
25         )
26
```

Comprobamos como el usuario JJJJ recién creado y logeado recibe el correo y se almacena en nuestro directorio var/www/correo.

```
*** uWSGI is running in multiple interpreter mode ***
spawned uWSGI worker 1 (and the only) (pid: 3212, cores: 1)
[pid: 3212|app: 0|req: 1/1] 79.116.43.46 () {54 vars in 939 bytes} [Wed Mar 8 20:07:36 2023] GET /account
ed 6220 bytes in 539 msecs (HTTP/2.0 200) 10 headers in 470 bytes (1 switches on core 0)
Añadiendo el usuario al grupo Blogger
[pid: 3212|app: 0|req: 2/2] 79.116.43.46 () {60 vars in 1108 bytes} [Wed Mar 8 20:07:40 2023] POST /acco
```



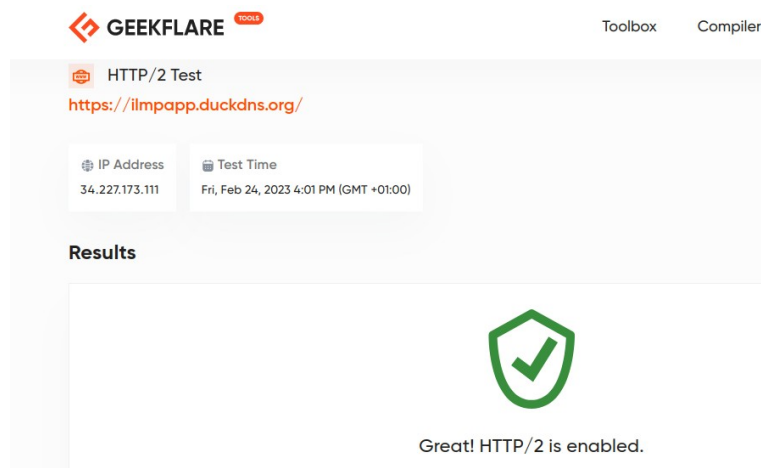
```
1 Content-Type: text/plain; charset="utf-8"
2 MIME-Version: 1.0
3 Content-Transfer-Encoding: 7bit
4 Subject: Subject
5 From: from@example.com
6 To: JJJJ@hotmail.com
7 Date: Wed, 08 Mar 2023 20:07:40 -0000
8 Message-ID:
```

Para añadir http2 nos dirigimos al fichero `sudo nano /etc/nginx/sites-enabled/default` y añadimos `http2` tras port 443 generado por certbot.

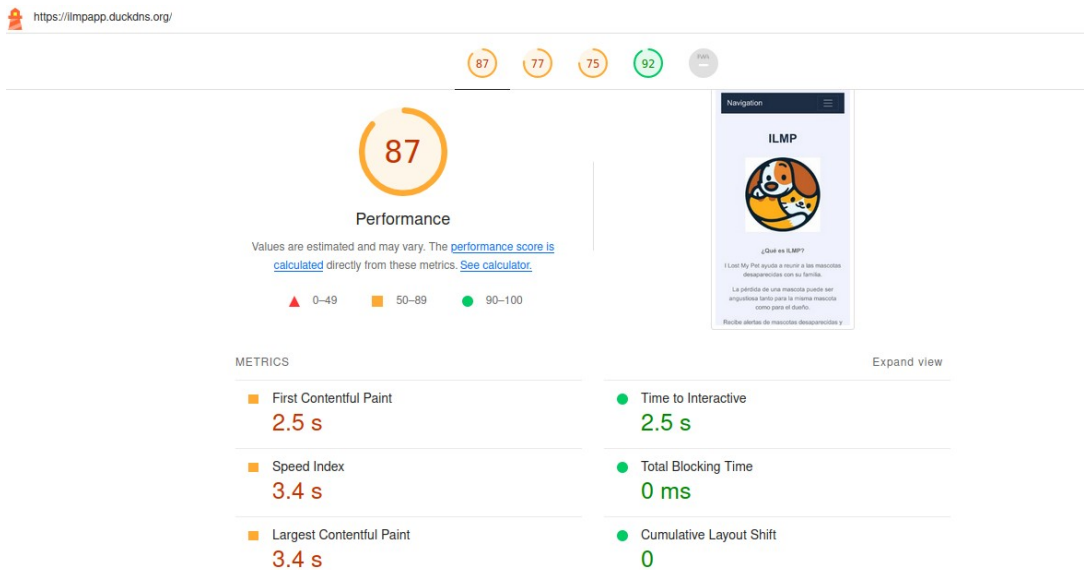
```
GNU nano 6.2 /etc/nginx/sites-enabled/default
    include /var/www/ilmp/uwsgi_params; # the uwsgi_
    }

listen 443 ssl http2; # managed by Certbot
```

Hecho esto realizamos un `systemctl restart` al servicio de `nginx` y un `collectstatic` a nuestra aplicación para aplicar los cambios.



Ahora realizaremos una comparativa con el antes y después de nuestra aplicación visualizandola con Lighthouse.



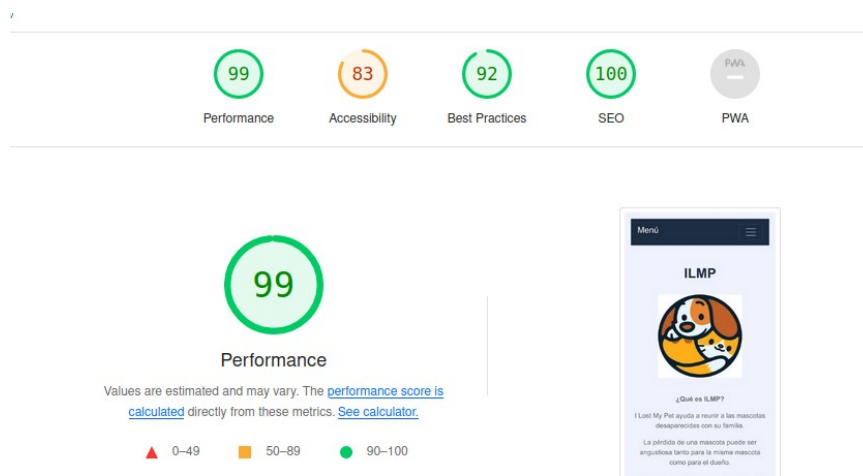
Ahora realizaremos los cambios pertinentes tales como, cambio de formato de imagen, reducción de uso innecesario del css, compresión del nginx, cache control y demás modificaciones entre otras para aumentar la puntuación.

Almacenamiento cache (2dias):

```
location / {
    expires 2d;
    add_header Cache-Control "public, no-transform";
```

Y en cabeceras:

Comprobación actual de puntuación la página de inicio tras los cambios:

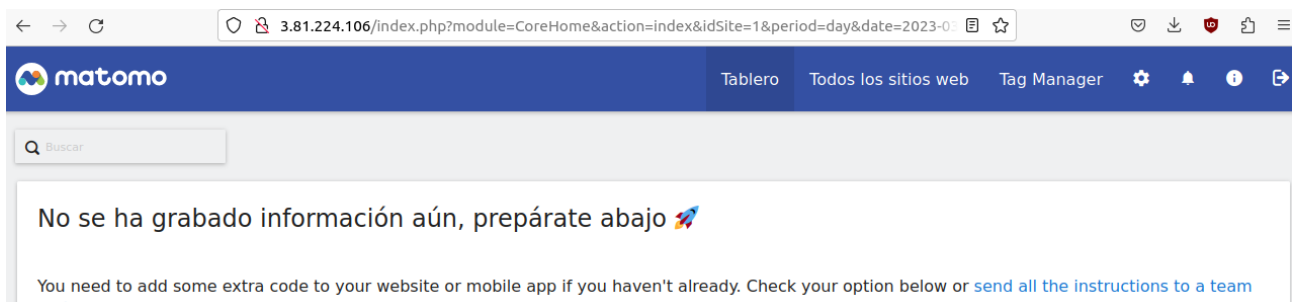


Para la analítica web usando matomo haremos uso de Docker de bitnami.

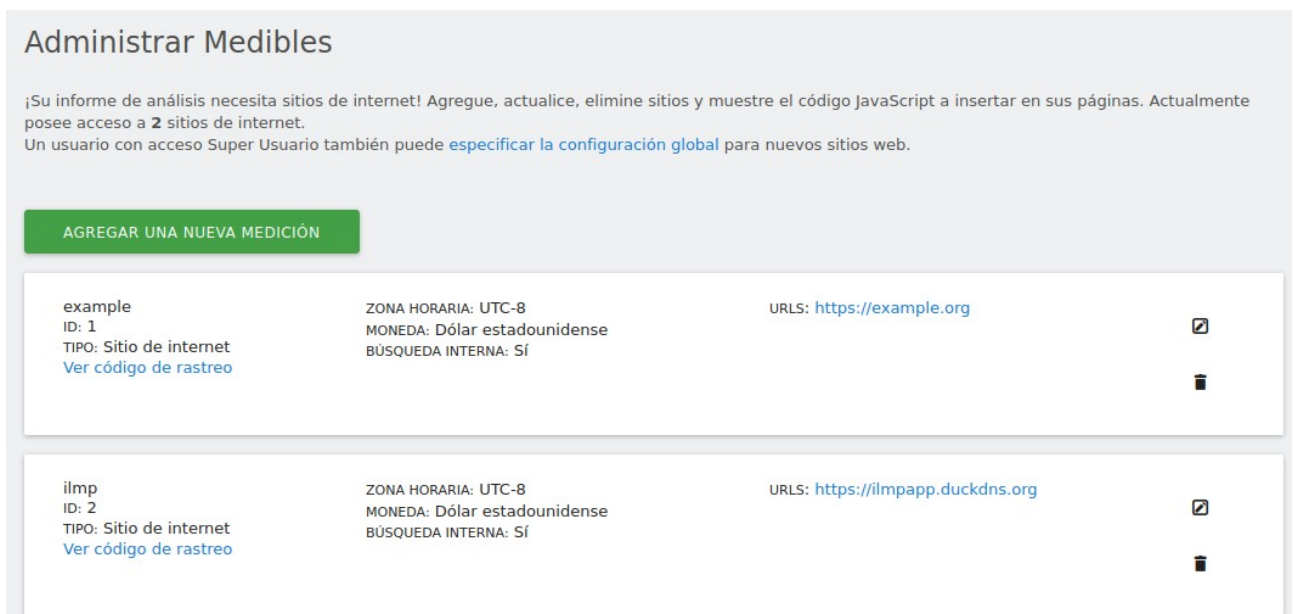
Para ello primero crearemos una nueva instancia EC2 en AWS, abriremos los puertos y realizaremos las configuraciones necesarias.

“docker ps -a”

Una vez levantado el docker accedemos escribiendo en el navegador la dirección de nuestra instancia seguido del puerto y accederemos haciendo uso de las credenciales User:bitnami.



Agregamos la dirección de nuestra aplicación para el analisis.



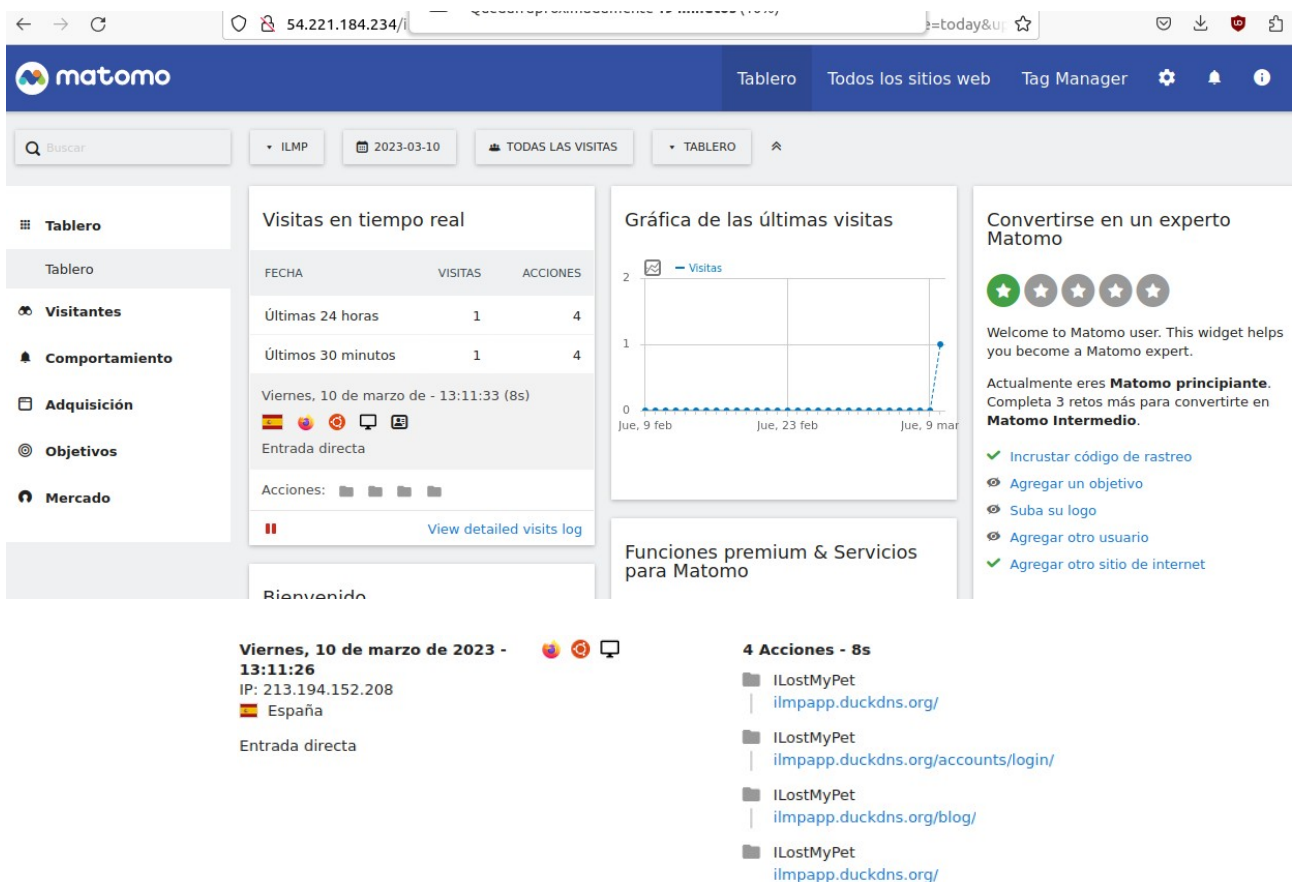


En nuestra instancia donde se ubica la aplicación agregamos en la cabecera de nuestros ficheros html la siguiente pieza de código para aquellas páginas que queramos analizar.

```
<!-- Matomo -->
<script>
  var _paq = window._paq = window._paq || [];
  /* tracker methods like "setCustomDimension" should be called before "trackPageView" */
  _paq.push(['trackPageView']);
  _paq.push(['enableLinkTracking']);
  (function() {
    var u="//3.81.224.106/";
    _paq.push(['setTrackerUrl', u+'matomo.php']);
    _paq.push(['setSiteId', '2']);
    var d=document, g=d.createElement('script'), s=d.getElementsByTagName('script')[0];
    g.async=true; g.src=u+'matomo.js'; s.parentNode.insertBefore(g,s);
  })();
</script>
<!-- End Matomo Code -->
```

Modificamos la dirección http:// y la ubicación del fichero a <https://ilmpapp.duckdns.org/static/js/matomo.js> y añadimos el fichero matomo.js a esa ubicación.

Hecho esto visualizamos nuestra página en matomo y vemos la gráfica.



## AWS AutoScaling...

EC2 > Grupos de Auto Scaling > Crear grupo de Auto Scaling

Paso 1  
Elija la plantilla de lanzamiento o la configuración

Paso 2  
Elegir las opciones de lanzamiento de instancias

Paso 3 - opcional  
Configurar las opciones avanzadas

Paso 4 - opcional  
Configurar políticas de escalado y tamaño de grupo

Paso 5 - opcional  
Añadir notificación

Paso 6 - opcional  
Añadir etiquetas


## Revisar Info

Paso 1: Elegir la plantilla de lanzamiento o la configuración Editar

### Detalles del grupo

Nombre del grupo de Auto Scaling  
Autoscaling

Plantilla de lanzamiento

|   |         |
|---|---------|
| Plantilla de lanzamiento  | Versión |
| <a href="#">Template2</a>  | Default |
| lt-014bcb773e0da12ef  |         |

Descripción

Paso 2: Elegir las opciones de lanzamiento de instancias Editar

Hasta el punto actual se ha realizado pequeñas modificaciones en la aplicación.