ANÁLISIS Y ARQUITECTURA DEL CÓDIGO – APP DJANGO

WILDDETECT

NOMBRE: JOSE LUIS MAYO CALVO

CURSO: 2º DAM

CENTRO: IES RIBERA DE CASTILLA

CONTENIDO DE LA MEMORIA

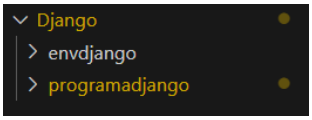
EXPLICACIÓN DEL FUNCIONAMIENTO DE UNA APP CON DJANGO

La página web realizada es una página para la detección de animales en los entornos salvajes de nuestra comunidad autónoma (Castilla y León). En esta aplicación tendremos varios apartados en el que el usuario podrá tomar fotos que serán analizadas por unos modelos de Inteligencia Artificial, que predecirán estas imágenes, además de numerosas funciones más de las que el usuario podrá disfrutar.

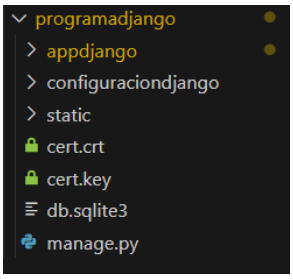
Dicha página, está configurada y creada para que el usuario no tenga ningún tipo de dificultad a la hora de utilizarla. Se caracteriza por ser una aplicación accesible, cómoda y que podrá usar personas de cualquier edad.

Está realizada con el lenguaje de programación de Python y el framework Django, además de usar Visual Studio Code como IDE.

Se compone de una carpeta que almacena todo el proyecto y dentro de ella, dos carpetas principales, una carpeta que almacena los archivos de configuración del proyecto y otra carpeta que contiene la aplicación (código de una aplicación completa dentro del proyecto).



Además, esta aplicación esta creada sobre un entorno virtual, con la finalidad de que las dependencias o configuraciones globales afecten de la menor manera posible al proyecto. A la hora de desplegar la aplicación, el haber creado un entorno virtual va a permitir que esta función sea mucho más sencilla en muchos servidores. También, con la creación de un entorno virtual “envdjango”, hacemos que el sistema sea mucho más limpio, ya que no hay problemas entre proyectos, ya que, la aplicación, está en un campo individual.

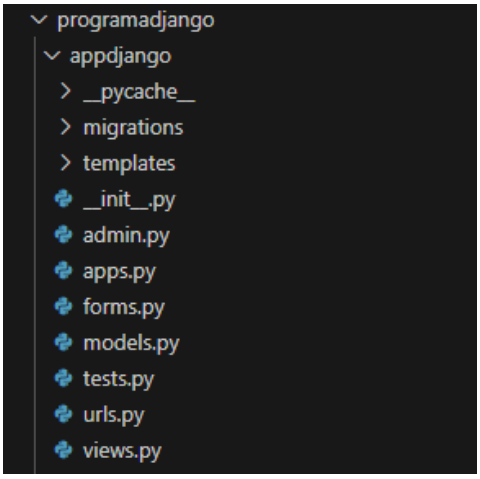


La carpeta donde se encuentran todas las configuraciones acerca del proyecto se llama “programadjango”. En ella nos encontramos dos carpetas en la que tenemos archivos de vital importancia para la creación y desarrollo de la aplicación web.

A su vez, esta carpeta se compone de dos subcarpetas, donde encontramos dichos archivos.

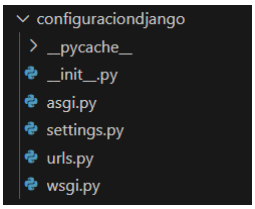
La primera carpeta, es la que contiene la aplicación. Se llama “appdjango” y es la carpeta donde se programa la página web, tanto la parte de back-end, como la de front-end.

Se compone de varios archivos de relativa importancia:



- Views.py: se encuentran todas las vistas que va a poseer la aplicación. Con vistas, nos referimos al código que va a dar funcionalidad a la parte visual de la aplicación, permitiendo loggear a los usuarios, mostrar mensajes de éxito o error, eliminar cuentas de la base de datos, crear nuevas…
- Urls.py: es el archivo donde se configura todas las rutas a las que la aplicación va a tener acceso. Estas rutas llevan asociadas unas vistas, haciendo que cuando se navegue a esa ruta, la vista se ejecute y se cree la funcionalidad de la app.
- Models.py: es el archivo que contiene el código para la creación de la base de datos. Aquí creamos los modelos (tablas) que vamos a usar en la base de datos. En nuestro caso, creamos la tabla Animales y Lugar con sus atributos correspondientes.
  Django da una ventaja a sus programadores haciendo que las tablas con mayor uso a la hora de programar vienen predeterminadas, por lo que la tabla Usuarios no hace falta crearla, únicamente nombrarla y acceder a los métodos que vienen predeterminados.
- Forms.py: archivo donde aparecen todos los formularios de la aplicación, los cuales, irán vinculados a los HTML que explicaremos posteriormente, para poder hacer un puente entre la parte del back-end y la del front-end.
  Aquí creamos el formulario de la página de inicio de sesión y el de la página de registro, para una mayor comodidad a la hora de programar esas partes de código.
- Tenemos la carpeta templates, en donde se guardan los archivos HTML, es decir, la configuración de la parte visual de la página.
  En esta carpeta, almacenamos varios HTML:
  - Login.html: archivo que programa la parte visual del login de usuario.
  - Registro.html: archivo que programa la parte visual del registro de usuario.
  - Inicio.html: página donde se reconocerán los animales, añadiendo el modelo entrenado.
  - Reconocimiento.html: página donde se muestran todos los reconocimientos almacenados en un mapa.
  - Ajustes.html: página donde se muestra la configuración de la parte visual de ajustes (cambiar contraseña, eliminar cuenta, visualización de los datos del usuario autenticado…).
  - Resultado.html: mostramos la detección de las IA a partir de la imagen subida en “reconocimiento.html”, pudiendo guardar el reconocimiento en la base de datos o por el contrario, reconocer otra imagen sin guardarla.
- Migrations: es la carpeta que almacena las últimas migraciones realizadas en la base de datos.
  Las migraciones es una parte fundamental para el correcto funcionamiento de cualquier base de datos (relacional o no relacional) al usar django.
  Son los archivos que reflejan los cambios del archivo models.py (modelos de la base de datos), en la base de datos, ya sea relacional o no relacional.

La segunda carpeta, se denomina “configuraciondjango”, donde tenemos:



- Settings.py: es el archivo principal de configuración del proyecto, en el que tenemos apartados como BASE_DIR (ruta base del proyecto), SECRET_KEY (clave secreta usada para firmar cookies, tokens…, ALLOWED_HOST (lista de dominios a los que les permites el acceso a la app), INSTALLED_APPS (define las apps que están activas en el proyecto), MIDDLEWARE (procesos que ocurren antes o después de una respuesta (request)), BASE DE DATOS (configuramos el archivo que va a ser la base de datos del sistema), entre muchos otros…
- Urls.py: contiene la ruta principal de la aplicación web.

Por último, fuera de ambas carpetas, pero dentro de la carpeta “programadjango”, tenemos:

- Manage.py: archivo que hay que ejecutar para que se pueda visualizar la aplicación, es decir, es el encargado de ejecutar el servidor de desarrollo.
- db.sqlite3: contiene el modelo de la base de datos en formato script, permitiendo el almacenamiento de información en su interior.
- Cert.key: contiene una clave privada para la aplicación que, usada junto al archivo “cert.crt”, podremos acceder a la página web cuando el navegador requiera tener el protocolo HTTPS para acceder a dichas páginas.
- Static: carpeta que contiene dos subcarpetas:
    - Img: subcarpeta donde se almacena las imágenes que van a ser usadas en nuestra página web
    - Styles: subcarpeta donde almacenamos los archivos CSS externos a los HTML configurados. Los CSS externos, están realizados para las configuraciones de varios archivos HTML de forma global, es decir, para los HTML que necesitan una misma forma de configuración, por ejemplo, en menús.

EXPLICACIÓN DEL CÓDIGO DE LA PÁGINA WEB

La parte que ha sido programada, es decir, todos los archivos en los que se encuentra la parte programada están en la carpeta appdjango.

```python
from django.db import models
from django.contrib.auth import get_user_model

#creamos la tabla Lugar con sus atributos correspondientes
class Lugar(models.Model):
    latitud = models.FloatField()
    longitud = models.FloatField()
    #obligamos al programa a que la tabla en la base de datos se llame "Lugar"
    class Meta:
        db_table = 'Lugar'

#creamos la tabla Animales con sus atributos correspondientes
class Animales(models.Model):
    user = get_user_model()

    usuario = models.ForeignKey(user, on_delete=models.CASCADE, related_name='animales')  # Relación many-to-one
    nombreAnimal = models.CharField(max_length=50)
    cantidad = models.IntegerField()
    lugarEncuentro = models.ForeignKey('Lugar', on_delete=models.CASCADE)  # Relación con la tabla Lugar
    fechaHora = models.DateTimeField(auto_now_add=True)
    #obligamos al programa a que la tabla en la base de datos se llame "Animales"
    class Meta:
        db_table = 'Animales'
```

Empezamos por el archivo de configuración de base de datos, “models.py”. En este archivo, modelamos las tablas que va a tener la base de datos de nuestro proyecto.

En este caso, vamos a tener dos tablas:

- Animales: es la tabla que va a almacenar los animales avistados. Esta tabla, contiene los atributos user (usuario que ha encontrado el animal), nombreAnimal (que almacenará el animal que ha encontrado), cantidad (en caso de avistar mas de 1 miembro), fechaHora (almacena la fecha y la hora en la que se ha encontrado ese animal y lugarEncuentro (guarda como FK el lugar en el que ha sido encontrado el animal).
- Lugar: es la tabla que se encarga de guardar los lugares en los que son encontrados los animales, con los atributos latitud y longitud. Esta tabla, tiene una referencia en la tabla Animales.
- User: esta tabla no aparece en el modelo, pero viene incorporada por Django, ya que, este framework, incorpora esta tabla de manera automática en todos los proyectos, debido a su habitual uso en casi todos los proyectos que se realizan.
  Esta tabla almacena los usuarios que han sido registrados en la aplicación, teniendo numerosos atributos capaces de darnos datos de:
  - Correo del usuario
  - Contraseña encriptada
  - Nombre de usuario
  - Si está autenticado en ese momento o no
  - Último momento de inicio de sesión
  - Si el usuario tiene permisos de superusuario o no

Entre otros, aunque, estos son para nosotros los más relevantes a priori.

ARCHIVO FORMS

```python
from django import forms
from django.core.validators import RegexValidator

#creacion del formulario del login para que el usuario pueda iniciar sesion
class LoginFormulario(forms.Form):
    correo = forms.EmailField(
        label="Correo electrónico",
        widget=forms.EmailInput(attrs={'class': 'form-control', 'placeholder': 'Tu correo electrónico'})
    )
    contrasena = forms.CharField(
        label="Contraseña",
        widget=forms.PasswordInput(attrs={'class': 'form-control', 'placeholder': 'Tu contraseña'})
    )

#creacion del formulario de registro para que el usuario pueda registrar una cuenta en la base de datos
class RegistroFormulario(forms.Form):
    nombre = forms.CharField(
        label="Nombre",
        widget=forms.TextInput(attrs={'class': 'form-control', 'placeholder': 'Tu nombre'})
    )
    correo = forms.EmailField(
        label="Correo electrónico",
        widget=forms.EmailInput(attrs={'class': 'form-control', 'placeholder': 'ejemplo@gmail.com'})
    )
    contrasena = forms.CharField(
        label="Contraseña",
        widget=forms.PasswordInput(attrs={'class': 'form-control', 'placeholder': 'Tu contraseña'}),
        validators=[RegexValidator(
            regex=r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d).{8,}$',
            message="La contraseña debe tener al menos 8 caracteres, una mayúscula, una minúscula y un número."
        )]
    )

    def clean_contrasena(self):
        contrasena = self.cleaned_data.get('contrasena')
        return contrasena
```

El archivo en donde se crean los formularios que hay en la aplicación. En mi caso, he introducido los formularios de las dos páginas iniciales, en los que se configura la cantidad de campos de texto a rellenar por el usuario, siendo posteriormente vinculados al HTML.

En loginFormulario (formulario de la página de inicio de sesión), se crean dos campos de texto:

- Correo: se introduce el correo electrónico, que lo vamos a denominar correo.
  Este campo está configurado como un campo de email, que va a hacer, que no se admitan textos sin un @ y algo puesto después.
- Contraseña: se introduce la contraseña del usuario ‘contrasena’.
  Este campo está configurado como un campo de contraseña, que va a hacer que se muestre la contraseña introducida mediante puntos, en vez de mostrar los caracteres, por cuestión de seguridad.

En registroFormulario (formulario de la página de registro), se crean tres campos de texto:

- Nombre: se introduce el nombre del usuario, lo vamos a denominar nombre.
  Este campo está configurado como un campo de texto, por lo que no tiene ninguna configuración especial.
- Correo: se introduce el correo electrónico, que lo vamos a denominar correo.
  Este campo está configurado como un campo de email, que va a hacer, que no se admitan textos sin un @ y algo puesto después.

Contraseña: se introduce la contraseña del usuario, que se llama “contrasena”, en la que configuramos que la contraseña debe de tener un mínimo de 8 caracteres, una mayúscula, una mínuscula y un número. Este campo está configurado como un campo de contraseña, que va a hacer que se muestre la contraseña introducida mediante puntos, en vez de mostrar los caracteres, por cuestión de seguridad.

Además, añadimos un método que recoge la contraseña y la devuelven. Es usada por distintas funciones a la hora de recoger la contraseña para validaciones, cambios de contraseña o gestiones en el usuario.

Los campos con la configuración placeholder, aparece dentro del campo de texto lo configurado cuando está vacío, con la finalidad de que aparezca una guía para el usuario.

ARCHIVO VISTA (VIEWS)

Seguimos con el archivo views.py:

Lo primero que tenemos es la parte de las importaciones de las distintas funciones utilizadas en el programa, que iremos analizando su uso de cada una según avance el documento.

```python
import base64
import io
from PIL import Image # type: ignore
from urllib import request
from django.http import JsonResponse
from django.shortcuts import render, redirect

from django.contrib.auth.hashers import check_password
from django.contrib.auth import login
from django.contrib.auth.hashers import make_password
from django.contrib.auth.models import User
from .forms import LoginFormulario, RegistroFormulario
from .models import Animales, Lugar
from django.contrib.auth.models import User
import json
from django.http import JsonResponse
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array
import numpy as np
from django.contrib.auth import update_session_auth_hash
from django.core.mail import send_mail
from django.shortcuts import render, redirect
from django.contrib import messages
from django.conf import settings
from django.contrib.auth import logout
```

Recordemos que en el archivo “views.py” se configuran todas las vistas que se van a utilizar en la aplicación. Es decir, hacemos que la parte del front-end (HTML), sea funcional y pueda realizar las funciones pertinentes.

```python
modeloReptiles = load_model(r"C:\Users\Jose\Desktop\TFG\Django\programadjango\appdjango\modelos\modeloReptiles.keras")
clases_modelo_reptiles = ['Culebra', 'Lagarto', 'Rana', 'Tortuga']
modeloDomestico = load_model(r"C:\Users\Jose\Desktop\TFG\Django\programadjango\appdjango\modelos\modeloDomestico.keras")
clases_modelo_domesticos = ['Bisonte', 'Burro', 'Caballo', 'Cabra', 'Cerdo', 'Cisne', 'Conejo', 'Gallina', 'Gato', 'Oveja', 'Pavo', 'Perro', 'Vaca']
modeloAves = load_model(r"C:\Users\Jose\Desktop\TFG\Django\programadjango\appdjango\modelos\modeloAve.keras")
clases_modelo_aves = ['Aguila', 'Buitre', 'Cigüeña', 'Ganso', 'Grulla', 'Halcon', 'Lechuza']
modeloSalvaje = load_model(r"C:\Users\Jose\Desktop\TFG\Django\programadjango\appdjango\modelos\modeloSalvaje.keras")
clases_modelo_salvajes = ['Ardilla', 'Ciervo', 'Jabali', 'Liebre', 'Lince', 'Lobo', 'Marmota', 'Nutria', 'Oso', 'Tejon', 'Zorro']
```

Primero, se configuran las rutas de los modelos de inteligencia artificial, es decir, se indica la ubicación del archivo que contiene el modelo entrenado capaz de identificar animales a partir de imágenes. Además, se definen las clases o categorías que el modelo es capaz de reconocer, colocadas en el mismo orden en que fueron utilizadas durante su entrenamiento. Esto es importante porque, al realizar una predicción, el modelo no devuelve directamente el nombre del animal, sino una lista de probabilidades asociadas a cada clase. Esas probabilidades están organizadas por índice, y ese índice corresponde a una clase específica. Por ejemplo, si el modelo devuelve una alta probabilidad en el índice 2, ese índice debe estar relacionado con la clase número 2 en nuestra lista (por ejemplo, "león"). Así, se puede traducir correctamente la salida numérica del modelo al nombre real del animal.

VISTA LOGIN

```python
#vista usada para el login de nuestra pagina web
def vistaLogin(request):
    if request.method == "POST":
        formulario = LoginFormulario(request.POST)
        if formulario.is_valid(): # si el formulario es valido, recogemos los datos en variables
            correo = formulario.cleaned_data['correo']
            contrasena = formulario.cleaned_data['contrasena']

            user = User.objects.filter(email=correo).first()

            if user is None or not check_password(contrasena, user.password):
                messages.error(request, "Credenciales inválidas.")
            else:
                login(request, user)  # Logeamos el usuario
                return redirect('inicio')  # Redirigir a la página de inicio

        else:
            messages.error(request, "Formulario no válido.")

    else:
        formulario = LoginFormulario()

    return render(request, "appdjango/login.html", {"form": formulario})
```

Es la vista donde se configura la funcionalidad del login, es decir, el inicio de sesión. En esta vista, vinculamos el formulario LoginFormulario del archivo “forms.py”, guardándolo en una variable. Si el formulario es válido, obtenemos los datos que ha metido el usuario en los campos y lo guardamos en variables.

Posteriormente, comprobamos que el correo del usuario ya exista en la base de datos. Si existe, y además, la contraseña es igual que la vinculada al correo en la base de datos, redirigimos al usuario a la página de inicio. Si la contraseña no coincide o el correo del usuario no existe, mostramos un mensaje de error.

La última línea del código se refiere a que esta vista está vinculada al archivo “login.html”, que es el archivo donde se configura la parte del front-end de la página web. Además, pasamos el formulario mediante la variable “form”, que la usaremos después en el archivo html para referirnos a este formulario y llegar a relacionar el HTML con la base de datos.

VISTA REGISTRO

```python
#vista para el registro de nuestra pagina web
def vistaRegistro(request):
    if request.method == "POST":
        formulario = RegistroFormulario(request.POST)
        if formulario.is_valid():
            #cogemos los datos que ha insertado el usuario a la hora de registrarse y lo guardamos en variables
            nombre = formulario.cleaned_data['nombre']
            correo = formulario.cleaned_data['correo']
            contrasena = formulario.cleaned_data['contrasena']

            if not re.match(r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d).{8,}$', contrasena): #verificamos si la contraseña cumple con los requisitos
                formulario.add_error('contrasena', 'La contraseña debe tener al menos 8 caracteres, una mayúscula, una minúscula y un número.')
                return render(request, "appdjango/registro.html", {"form": formulario})
            else:
                if User.objects.filter(email=correo).exists(): #verificamos que el correo no esté registrado ya en la base de datos
                    formulario.add_error('correo', 'Este correo ya está registrado.')
                    return render(request, "appdjango/registro.html", {"form": formulario})
                else:
                    if User.objects.filter(username=nombre).exists(): #verificamos que el nombre no esté registrado ya en la base de datos
                        formulario.add_error('nombre', 'Este nombre ya está registrado.')
                        return render(request, "appdjango/registro.html", {"form": formulario})
                    else:
                        #si el usuario no existe, encriptamos la contraseña y creamos un nuevo usuario insertandolo en la base de datos
                        contrasena_encriptada = make_password(contrasena)
                        nuevo_usuario = User(username=nombre, email = correo, password = contrasena_encriptada)
                        nuevo_usuario.save()
                        #una vez insertado, mandamos al usuario a logearse
                        return redirect('login')

    else:
        formulario = RegistroFormulario()
    return render(request, "appdjango/registro.html", {"form": formulario})
```

Es la vista donde se configura el registro de la página web, es decir, donde los usuarios pueden acceder a registrarse en la base de datos mediante sus datos personales. La forma de programacion es la misma que la pantalla anterior (inicio sesión).

Guardamos el formulario en una variable y comprobamos si es valido. Si es así, guardamos los datos introducidos por el usuario en los campos de texto en variables y comprobamos tres cosas: que la contraseña cumpla los requisitos, es decir, que tenga un mínimo de 8 caracteres, una letra mayúscula, una minúscula y un número, que el correo no haya sido introducido anteriormente, y que el nombre de usuario no haya sido introducido por otro usuario, ya que no se puede iniciar sesion dos veces con la misma cuenta ni con el nombre que haya utilizado otro usuario en otra cuenta.

Si la cuenta existe en la base de datos, la contraseña no es válida o el nombre está repetido, mostramos un mensaje de error al usuario para que vuelva a introducir otra vez el correo o introduzca uno nuevo. En caso contrario, encriptamos la contraseña, creamos un usuario con los datos introducidos y la contraseña encriptada, y lo guardamos en la base de datos, redirigiendo al usuario a la pantalla de inicio de sesión (la pantalla anterior), para que el usuario se loggeé en la página.

Al igual que en la pantalla anterior, vinculamos la vista con la pagina de registro del HTML y pasamos el formulario mediante la variable “form”, que usaremos en el HTML para referirnos a esta vista.

VISTA INICIO

```python
#vista para la pagina de inicio (despues del login)
def vistaInicio(request):
    return render(request, 'appdjango/inicio.html')
```

Definimos la vista de la pantalla de inicio (la pantalla que aparece después del login).

En esta vista no hay ninguna configuracion, únicamente redirigimos al HTML donde se define la vista de esta página.

VISTA RECONOCIMIENTOS

```python
#vista para el historial de reconocimientos de animales
def vistaReconocimientos(request):
    user = request.user

    # Obtener todos los animales almacenados en la base de datos
    animales = Animales.objects.filter(usuario = user)

    return render(request, 'appdjango/reconocimiento.html', {'animales': animales})
```

Seguimos con la vista de la pantalla de reconocimientos. Pantalla donde se muestra los animales reconocidos por el usuario, así como el lugar donde se han reconocido mediante un mapa adjunto a la página web.

En esta vista, decimos que coja los datos del usuario que ha iniciado sesion, guardandolos en una variable, y haciendo que se devuelvan los animales guardados en la base de datos con este usuario, con la finalidad de que unicamente se muestren en el mapa interactivo los animales encontrados por el propio usuario.

Por último, como en todas las vistas, vinculamos la vista con el archivo HTML, pasando la varibale que nos interesa, en este caso, los animales encontrados con el id del usuario que ha iniciado sesion.

VISTA AJUSTES

```python
#vista para los ajustes de la pagina web
def vistaAjustes(request):
    return render(request, 'appdjango/ajustes.html', {'usuario': request.user, 'email': request.user.email, 'contrasena': request.user.password} )
```

A continuacion, configuramos la vista de ajustes, en la que únicamente vinculamos la vista con el archivo HTML, y pasamos a este HTML las variables que tienen el nombre, email y la contraseña del usuario con la finalidad de que aparezca informacion del usuario loggeado en esta pantalla, y, además, el usuario pueda cambiar su contraseña actual por otra nueva.

VISTA ELIMINAR CUENTA

```python
#funcion para eliminar la cuenta del usuario que ha iniciado sesion
def eliminarCuenta(request):
    if request.user.is_authenticated:  # Verifica si el usuario está autenticado
        user = request.user
        user.delete()  # Elimina el usuario de la base de datos
        return JsonResponse({"status": "success"})
    else:
        return JsonResponse({"status": "error", "message": "No estás autenticado"})
```

La vista eliminar cuenta, que la usamos para cuando el usuario, en la página de ajustes, pulsa el boton de eliminar cuenta. En esta vista, hacemos que si el usuario está autenticado, obtengamos los datos del mismo, y los borremos de la base de datos, borrando también, la informacion de otras tablas que tenga que ver con su usuario. Si el usuario no está autenticado, no se podrá realizar la accion, por cuestion de seguridad y evitar que se eliminen cuenta de forma no deseada.

VISTA CAMBIAR CONTRASEÑA

```python
#funcion para cambiar la contraseña del usuario que ha iniciado sesion
def cambiar_contrasena(request):
    if request.method == "POST":
        data = json.loads(request.body)
        actual = data.get("actual") #obtenemos el valor de la contraseña actual y la nueva
        nueva = data.get("nueva")

        usuario = request.user
        if not usuario.check_password(actual): #comprobamos que la contraseña actual sea correcta
            return JsonResponse({"success": False, "error": "La contraseña actual es incorrecta"}, status=400)

        usuario.set_password(nueva) #introducimos la contraseña nueva al usuario
        usuario.save() # guardamos los cambios
        update_session_auth_hash(request, usuario)  # Evita que el usuario cierre sesión

        return JsonResponse({"success": True})

    return JsonResponse({"success": False, "error": "Método no permitido"}, status=405)
```

Creamos la vista cambiar contraseña, en la que el usuario podrá cambiar su contraseña actual por otra que desee. Para ello tendrá que introducir la contraseña actual y dos veces la contraseña nueva, por cuestiones de seguridad y protección del usuario.

Primero, obtenemos los valores de la contraseña actual, la contraseña nueva y el usuario actual que está loggeado. Si el valor de la contraseña actual no coincide con la del usuario loggeado, se muestra un mensaje de error. En caso contrario, actualizamos la contraseña del usuario a la contraseña nueva introducida.

Normalmente, al introducir una nueva contraseña, la página se dirigirá a la pantalla inicial de nuevo, en la que el usuario tendrá que iniciar sesión. Para ello, lo evitamos con la línea de código de “update_session”. Una vez realizado todo esto, devolvemos una respuesta JSON válida.

VISTA DETECCIÓN RESULTADO

```python
def deteccion_resultado(request):
    if request.method == 'POST':
        if 'archivo' in request.FILES:
            imagen = Image.open(request.FILES['archivo'])
        elif request.POST.get('fotobase64'):
            base64_str = request.POST.get('fotobase64').split(',')[1]
            image_data = base64.b64decode(base64_str)
            imagen = Image.open(io.BytesIO(image_data))
        else:
            return render(request, 'appdjango/resultado.html', {'error':'No se recibió ninguna imagen'})
```

Seguimos con la vista detección resultado, que se encargará de usar los modelos de IA creados en la herramienta Jupyter con la finalidad de predecir el animal que exista en una imagen.

Para ello, enviamos una petición “POST” al formulario para ver si el usuario está enviando datos desde el formulario o no.

Primero, comprobamos que el usuario realiza una de estas dos acciones: realiza una foto en la misma aplicación (base64) o adjunta una imagen en el seleccionable habilitado en la página web. Además, cuando el usuario toma una foto en la aplicación, la imagen se decodifica y se carga como una imagen, almacenándola en la variable. En caso de no realizar ninguna acción, salta un error al usuario.

Cuando la imagen es válida, seguimos con el proceso:

```python
# Preprocesamiento
imagen = imagen.resize((128, 128))
imagen_array = img_to_array(imagen) / 128.0
imagen_array = np.expand_dims(imagen_array, axis=0)
```

Primero, codificamos la imagen con un tamaño de 128x128 y la convertimos a un array NumPy, la normalizamos y modificamos el modelo para que únicamente espere una imagen, es decir, configuramos el batch size para que tenga el tamaño de 1 imagen.

Una vez hecho esto, generamos una lista de probabilidades de que el animal incluido en la imagen sea el de la clase correspondiente y seleccionamos la clase con un índice de mayor probabilidad. Por último, calculamos la probabilidad en porcentaje de que el animal predicho sea ese y guardamos la clase elegida.

```python
# Evaluar modelo de reptiles
pred_reptil = modeloReptiles.predict(imagen_array)
prob_reptil = np.max(pred_reptil)
clase_reptil = clases_modelo_reptiles[np.argmax(pred_reptil)]

if prob_reptil >= 0.6:
    return render(request, 'appdjango/resultado.html', {
        'clase': clase_reptil,
        'probabilidad': round(prob_reptil * 100, 2)
    })
```

En caso de que la probabilidad de que el animal de la foto sea el elegido por la IA de animales reptiles no sea mayor o igual a un 60%, el modelo no mostrará ningún resultado en la página web, y pasaremos a reconocer con el modelo de animales domésticos, siguiendo la misma dinámica que en el anterior modelo.

```python
# Evaluar modelo doméstico
pred_domestico = modeloDomestico.predict(imagen_array)
prob_domestico = np.max(pred_domestico)
clase_domestico = clases_modelo_domesticos[np.argmax(pred_domestico)]

if prob_domestico >= 0.6:
    return render(request, 'appdjango/resultado.html', {
        'clase': clase_domestico,
        'probabilidad': round(prob_domestico * 100, 2)
    })
```

Hacemos una predicción de todas las clases, y elegimos la que tenga mayor probabilidad, guardando dicha clase. Si la probabilidad no es mayor o igual al 60%, pasaríamos al modelo de aves y así sucesivamente hasta completar todos los modelos, siendo el modelo de animales salvajes el último para predecir.

En caso de que un modelo reconozca un animal

```python
# Evaluar modelo salvajes
pred_salvajes = modeloSalvaje.predict(imagen_array)
prob_salvaje = np.max(pred_salvajes)
clase_salvaje = clases_modelo_salvajes[np.argmax(pred_salvajes)]

if prob_salvaje >= 0.6:
    return render(request, 'appdjango/resultado.html', {
        'clase': clase_salvaje,
        'probabilidad': round(prob_salvaje * 100, 2)
    })
```

```python
# Evaluar modelo aves
pred_aves = modeloAves.predict(imagen_array)
prob_aves = np.max(pred_aves)
clase_ave = clases_modelo_aves[np.argmax(pred_aves)]

if prob_aves >= 0.6:
    return render(request, 'appdjango/resultado.html', {
        'clase': clase_ave,
        'probabilidad': round(prob_aves * 100, 2)
    })
```

con una probabilidad del 60% o más, se enviará a la página de “resultado.html” la clase seleccionada, así como la probabilidad de acierto para mostrarla al usuario.

En caso contrario, cuando los modelos no reconozcan ningún animal con ningún modelo con más del 60% de probabilidad, la página web mostrará un mensaje de que no se ha podido detectar ningún animal.

Además, en caso de que el archivo adjuntado por el usuario fuese erróneo, el sistema mostrará este error en “resultado.html”:

```python
return render(request, 'appdjango/resultado.html', {'error': 'Método no válido'})
```

VISTA OBTENER COORDENADAS EXIF

```python
def obtener_coordenadas_exif(archivo_imagen):
    # Abrimos la imagen para leer los datos EXIF
    imagen = Image.open(archivo_imagen)
    # Cogemos los metadatos EXIF
    exif_data = imagen._getexif()
    # Si existen metadatos EXIF
    if exif_data:
        # Obtenemos la coordenada GPS
        for tag, value in exif_data.items():
            # Encontrar la latitud y longitud en los metadatos EXIF (corresponde con la etiqueta 34853)
            if tag == 34853:
                lat = value[2][0] / value[2][1]  # Grados latitud (2)
                lon = value[4][0] / value[4][1]  # Grados longitud (4)
                return lat, lon
    return None, None  # Si no se encuentra información EXIF, devolvemos nulo
```

Seguimos con la vista para obtener las coordenadas de una foto ya realizada. Primero obtenemos la imagen para leer los datos EXIF (datos de la imagen que contienen el punto GPS donde se realizó). Una vez obtenidos, miramos a ver si existen estos datos. Si existen, obtenemos las coordenadas GPS, entrando en la etiqueta necesaria para obtener dichas coordenadas. Si existen coordenadas, cogemos la latitud y la longitud. En caso contrario, devolvemos nulo.

VISTA GUARDAR ANIMAL

```python
def guardar_animal_BBDD(request):
    if request.method == 'POST':
        if request.content_type == 'application/json':
            try:
                data = json.loads(request.body)
                clase = data.get('clase')
                probabilidad = data.get('probabilidad')
                latitud = data.get('latitud')
                longitud = data.get('longitud')
            except json.JSONDecodeError:
                return JsonResponse({'success': False, 'error': 'Error en el formato JSON'}, status=400)
        else:
            clase = request.POST.get('clase')
            probabilidad = request.POST.get('probabilidad')
            latitud = request.POST.get('latitud')
            longitud = request.POST.get('longitud')

        if not latitud or not longitud:
            return JsonResponse({'success': False, 'error': 'Coordenadas faltantes'}, status=400)

        # Consulta si ya existe un lugar con las mismas coordenadas
        lugar_existente = Lugar.objects.filter(latitud=float(latitud), longitud=float(longitud)).first()

        if lugar_existente:
            lugar = lugar_existente  # Si existe, lo reutilizamos
        else:
            lugar = Lugar(latitud=float(latitud), longitud=float(longitud))  # Si no existe, lo creamos
            lugar.save()

        animal = Animales(usuario=request.user, nombreAnimal=clase, cantidad=1, lugarEncuentro=lugar)
        animal.save()

        return redirect('reconocimientos')

    return JsonResponse({'success': False, 'error': 'Método no válido'}, status=405)
```

Pasamos al código que guarda el animal reconocido en la base de datos. En la página de “resultado.html”, si pulsamos el botón de “Guardar Animal”, se nos ejecutará este código.

Primero, verificamos que los datos vengan de una petición POST. Si es así, verificamos que los resultados que se muestran van a ser en formato JSON. Al ser así también, obtenemos en variables la clase elegida, la probabilidad de la precisión de la IA y la latitud y longitud del usuario en el momento de tomar la foto, o de la misma foto.

Además, mediante la variable “data”, obtenemos los datos obtenidos en formato JSON de la petición.

En caso de haber algún error a la hora de recibir el archivo JSON con la información, se devolverá un mensaje de información.

Por otro lado, si el formulario no utiliza formato JSON, se asume que los datos vienen por formulario tradicional, por lo que recogeremos de la misma forma dichos datos.

Comprobamos que los datos de latitud y longitud no vienen vacíos, es decir, si se han recibido de manera correcta. En caso de tener dichos datos, verificamos si el lugar recibido ya está guardado en la base de datos, para evitar guardar muchas veces el mismo dato en la base de datos y así tener un cúmulo de información innecesario.

Si es así, obtenemos el lugar en una variable. En caso contrario, almacenamos el lugar en la base de datos y lo utilizamos para almacenar el animal en su tabla de la base de datos configurada.

Una vez creado el nuevo animal, redirigimos al usuario a la página de reconocimientos, para que el usuario pueda visualizar en el mapa el animal encontrado.

VISTA RECUPERAR CONTRASEÑA

```python
def recuperar_contrasena(request):
    if request.method == 'POST':
        email = request.POST.get('email_recuperacion')

        asunto = 'Recuperación de contraseña'
        # Mensaje en texto plano (para clientes que no soportan HTML)
        mensaje = 'Haz clic en el siguiente enlace para cambiar tu contraseña:\n\n'
        mensaje += 'http://127.0.0.1:8000/cambio_contrasena/'

        # Mensaje en formato HTML
        mensaje_html = '''
            <html>
                <body>
                    <p>Haz clic en el siguiente enlace para cambiar tu contraseña:</p>
                    <a href="http://127.0.0.1:8000/cambio_contrasena/">Pincha aquí para cambiar tu contraseña</a>
                </body>
            </html>
        '''

        try:
            send_mail(
                asunto,
                mensaje,  # Mensaje en texto plano
                settings.EMAIL_HOST_USER,
                [email],
                fail_silently=False,
                html_message=mensaje_html  # Enviar el mensaje como HTML
            )
            request.session['email_usuario'] = email
            request.session.modified = True
            messages.success(request, 'Se ha enviado un enlace a tu correo.')
        except Exception as e:
            messages.error(request, f'Error al enviar el correo: {str(e)}')

        return redirect('login')  # Redirigir al login

    return redirect('login')
```

Seguimos con la vista que se dedica a mandar un Gmail al usuario, cuando este a olvidado la contraseña y necesita recuperarla.

Como en las demás vistas, verificamos si la vista realiza una petición “POST”. En caso afirmativo, recogemos el email que ha usado el usuario en el formulario para que sea posible el envío de dicho mensaje.

Configuramos el asunto y el cuerpo del mensaje, ya sea para formato HTML y para navegadores que no soportan dicho formato, así, como la URL de la página a la que el usuario debe acceder para poder restaurar su contraseña.

Una vez hecho esto, procedemos al envío del correo con el asunto configurado, el mensaje en texto plano (no el HTML), el email configurado en el archivo “settings.py” para enviar dicho mensaje, el correo de destino (usuario), si ocurre un error lanza una excepción y por último, configuramos el correo para que se visualice en formato HTML.

Cuando se envía, guardamos el correo en la sesión para poder utilizarlo en la vista el cambio de contraseña y mostramos un mensaje de éxito.

En caso de surgir cualquier problema o error en el envío del correo electrónico, nos saltará un error. Por último, redirigimos al usuario a la página de inicio de sesión, al igual que si la petición enviada no sea POST.

VISTA CAMBIAR CONTRASEÑA

```python
def cambio_contrasena(request):
    return render(request, 'appdjango/cambioContraseña.html')
```

Seguimos con la vista que se encarga de mandar al usuario a la página

configurada para cambiar la contraseña del usuario, usada para cuando se envía el enlace por Gmail al usuario.

VISTA REESTABLECER CONTRASEÑA

```python
def reestablecer_contrasena(request):
    if request.method == 'POST':
        password1 = request.POST.get('password1')
        password2 = request.POST.get('password2')
        email = request.POST.get('email') or request.GET.get('email') or request.session.get('email_usuario')

        if not email:
            messages.error(request, 'No se encontró el correo del usuario.')
            return redirect('recuperar_contrasena')

        if password1 != password2:
            messages.error(request, 'Las contraseñas no coinciden')
            return redirect('reestablecer_contrasena')

        if len(password1) < 8:
            messages.error(request, 'La contraseña debe tener al menos 8 caracteres.')
            return redirect('reestablecer_contrasena')

        if not re.search(r'[A-Z]', password1):
            messages.error(request, 'La contraseña debe contener al menos una letra mayúscula.')
            return redirect('reestablecer_contrasena')

        if not re.search(r'[a-z]', password1):
            messages.error(request, 'La contraseña debe contener al menos una letra minúscula.')
            return redirect('reestablecer_contrasena')

        if not re.search(r'\d', password1):
            messages.error(request, 'La contraseña debe contener al menos un número.')
            return redirect('reestablecer_contrasena')

        try:
            usuario = User.objects.get(email=email)
            usuario.set_password(password1)
            usuario.save()
            logout(request)
            messages.success(request, 'Contraseña actualizada correctamente.')
            return redirect('login')
        except User.DoesNotExist:
            messages.error(request, 'Usuario no encontrado')
            return redirect('recuperar_contrasena')

    email = request.GET.get('email') or request.session.get('email_usuario')
    return render(request, 'appdjango/cambioContraseña.html', {'email': email})
```

Por último, vemos la configuración de la vista que configura la nueva contraseña del usuario en la página de cambio de contraseña. Esta vista, recoge la contraseña puesta por el usuario en el primer y segundo campo de texto, al igual que el correo recogido en la vista configurada cunado el usuario pulsa el botón “Enviar enlace”, donde recogimos el correo electrónico puesto por el usuario en ese formulario.

Una vez hecho esto, se vienen una serie de comprobaciones:

- El correo pasado existe (la variable no está vacía)
- Las contraseñas del campo 1 y 2 son iguales
- La contraseña tiene 8 o más caracteres
- La contraseña tiene al menos una letra mayúscula
- La contraseña tiene al menos una letra minúscula
- La contraseña tiene al menos un número

En caso de que alguna comprobación sea errónea, se mostrará un error al usuario.

Una vez hecho esto, cogemos el usuario de la base de datos en el que el correo registrado sea el mismo que el correo al que se ha enviado el mensaje para restaurar su contraseña. A ese usuario, le modificamos la contraseña, poniéndole la nueva y guardando dichos cambios. Redirigimos al usuario a la pestaña de inicio de sesión y mostramos un mensaje de éxito en esa misma pantalla.

Si el usuario del correo no existe, redirigimos al usuario a la pantalla de inicio de sesión y mostramos un mensaje de error en esa misma pantalla.

En caso de que el envío del formulario falle (porque no sea un método POST), obtenemos el usuario del email, primero de la URL a la hora de cambiar a la pantalla de cambio de contraseña. Si no se encuentra, se obtiene el correo de la sesión, ya que ha sido guardado anteriormente y redirigimos al usuario a la pantalla de cambio de contraseña, para que pueda rellenar dicho formulario.

ARCHIVOS HTML

Después de analizar todo el código del archivo "views.py", pasamos a analizar los HTML, que se encuentran dentro de la carpeta templates y dentro de la subcarpeta appdjango. Aquí, encontramos todos los archivos de configuración de las distintas páginas que hay en la página web.

INICIAR SESION

Empezamos con el archivo "login.html", que contiene la configuración del front-end de la página. En este archivo, tenemos la configuración CSS incorporada, entre las etiquetas head y style, con la finalidad de tener una mejor organización de todos los archivos, ya que, la cantidad de estos en los proyectos Django, son grandes.

Analicemos el código HTML y JavaScript utilizados en el archivo:
Dentro de la etiqueta body, se crea el código donde se modela la página:

```
<div class="container">
    <h2>INICIAR SESIÓN</h2>
    {% if form.errors %}
        <div class="alert alert-error">
            {% for field, errors in form.errors.items %}
                {% for error in errors %}
                    <p>{{ error }}</p>
                {% endfor %}
            {% endfor %}
        </div>
    {% endif %}

    {% if messages %}
        <div class="alert">
            {% for message in messages %}
                <div class="alert-{{ message.tags }}">
                    {{ message }}
                </div>
            {% endfor %}
        </div>
    {% endif %}
```

Creamos un contenedor, donde ponemos el título "INICIAR SESION".

Configuramos los errores:

Decimos que, si el formulario tiene algún error, recorremos cada campo (field) que contiene un error y la lista de errores (errors) que contiene ese campo. Mostramos los errores.

Justo debajo, configuramos los mensajes de información, de éxito o de advertencia, de la misma manera que los errores anteriores.

Si hay mensajes en el formulario que mostrar, recorremos los mensajes a mostrar y les mostramos uno a uno.

Aquí, modelamos el formulario que va a rellenar el usuario, vinculándolo con el creado en el archivo "forms.py", explicado anteriormente, usando la variable form que pasábamos en la vista del login en el archivo "views.py".

Para ello, creamos el formulario, enviando los datos mediante el método POST, evitando, así, que se muestren los datos en la URL, indicando que dicho formulario se enviará a la URL 'login', definida en el archivo "urls.py".

Creamos el campo de correo, renderizando automáticamente los creados en los archivos comentados anteriormente. De esta misma manera, creamos el campo de contraseña,

```
<form method="POST" action="{% url 'login' %}">
    {% csrf_token %}
    <div class="form-group">
        {{ form.correo.label_tag }}
        {{ form.correo }}
    </div>

    <div class="form-group">
        {{ form.contrasena.label_tag }}
        <div style="position: relative;">
            {{ form.contrasena }}
            <span id="eye-icon">&#128065;</span>
        </div>
    </div>

    <button type="submit">Iniciar Sesión</button>
</form>
```

incorporando el icono del ojo abierto (con la finalidad de que se pueda ver la contraseña en caracteres y de manera segura con los puntos).

Creamos el botón para recuperar la contraseña, en el que no cambiamos de pantalla, sino que definimos un id en el que configuraremos posteriormente lo que se realizará cuando se pulse dicho botón.

```
<!-- Enlace para recuperar contraseña -->
<div class="registrar_link">
    <p><a href="#" id="forgot-password-link">¿Olvidaste tu contraseña?</a></p>
</div>
```

```
<!-- Contenedor oculto para recuperar contraseña -->
<div id="forgot-password-container" style="display: none; margin-top: 20px;">
    <h3 style="text-align: center;">Recuperar contraseña</h3>
    <form method="POST" action="{% url 'recuperar_contrasena' %}">
        {% csrf_token %}
        <div class="form-group">
            <label for="email_recuperacion">Correo electrónico</label>
            <input type="email" id="email_recuperacion" name="email_recuperacion" required>
        </div>
        <button type="submit" id="btn_enviarenlace" name="btn_enviarenlace">Enviar enlace</button>
    </form>
</div>
```

Creamos un contenedor (será el que se desplegará a la hora de pulsar el botón de recuperación de contraseña anterior), en el que pondremos un titulo alineado en el centro de dicho contenedor y un formulario que se enviará a la URL ‘recuperar_contrasena’ mediante el método POST. En dicho formulario, crearemos un campo de texto para que el usuario introduzca su correo electrónico, con el campo ‘required’ (es obligatorio que rellene este campo), y un botón “Enviar enlace”.

Por último, creamos el botón de iniciar sesión.

```
<div class="registrar_link">
    <p>¿No tienes cuenta? <a href="{% url 'registro' %}">Regístrate</a></p>
</div>
```

Creamos otro contenedor, en el que introducimos un texto y un texto subrayado, que hacemos que si se pulsa, nos envíe a la página de registro.

```
const passwordInput = document.querySelector('#id_contrasena');
const eyeIcon = document.getElementById('eye-icon');

eyeIcon.addEventListener('click', function() {
    if (passwordInput.type === 'password') {
        passwordInput.type = 'text';
        eyeIcon.innerHTML = '&#128065;&#10060;';
    } else {
        passwordInput.type = 'password';
        eyeIcon.innerHTML = '&#128065;';
    }
});
```

Entre las etiquetas script, introducimos el código JavaScript, que es el código que permite dar funcionalidad en un archivo HTML. En este archivo, configuramos la funcion de abrir y cerrar el ojo, para poder ver la contraseña con los caracteres.

Primero, obtenemos el id de la contraseña y seleccionamos el icono del ojo, para saber si este está abierto o cerrado.

Creamos una funcion que alterne el ojo abierto y el ojo cerrado.

Si la contraseña está con los puntos, lo cambia a texto y cambia el icono de ojo abierto a ojo cerrado al pulsar. En caso contrario, se oculta la contraseña y se cambia a ojo abierto.

```
const olvidarContraseñaLink = document.getElementById('forgot-password-link');
const olvidarContraseñaContenedor = document.getElementById('forgot-password-container');

olvidarContraseñaLink.addEventListener('click', function(event) {
    event.preventDefault();
    olvidarContraseñaContenedor.style.display =
        olvidarContraseñaContenedor.style.display === 'none' ? 'block' : 'none';
});
```

Este código sirve para mostrar o dejar de mostrar un contenedor.

Guardamos en una variable el botón de “¿Olvidaste tu contraseña?” y el contenedor que queremos que aparezca a la hora de pulsar dicho botón.

Si pulsamos el botón, hacemos que si el contenedor esta oculto, lo muestra, y si este se esta mostrando, lo oculta. La etiqueta “event.preventDefault()”, evita que la página se refresque siempre que se pulse el botón, ya que unicamente queremos que aparezca el contendor nuevo.

REGISTRO

A continuación, seguimos con el archivo “registro.html”, en donde, al igual que todos los archivos HTML, está configurado el CSS entre las etiquetas “head” y “style”.

```
<div class="container">
    <h2>REGISTRO</h2>
    {% if form.contrasena.errors %}
        <div class="error">
            {% for error in form.contrasena.errors %}
                <p>{{ error }}</p>
            {% endfor %}
        </div>
    {% endif %}
    {% if form.nombre.errors %}
        <div class="error">
            {% for error in form.nombre.errors %}
                <p>{{ error }}</p>
            {% endfor %}
        </div>
    {% endif %}
    {% if form.correo.errors %}
        <div class="error">
            {% for error in form.correo.errors %}
                <p>{{ error }}</p>
            {% endfor %}
        </div>
    {% endif %}
```

Creamos un contenedor con el título “REGISTRO”, en el que justo debajo configuramos 3 tipos de mensajes distintos (contraseña, nombre y correo), que aparecerán en caso de que existan.

Estos mensajes están configurados para dar errores a la hora de rellenar el formulario, para permitir que el usuario corrija dichos errores y pueda llegar a registrarse de manera correcta.

```
<form method="POST">
    {% csrf_token %}

    <div class="form-group">
        {{ form.nombre.label_tag }}
        {{ form.nombre }}
    </div>

    <div class="form-group">
        {{ form.correo.label_tag }}
        {{ form.correo }}
    </div>

    <div class="form-group">
        {{ form.contrasena.label_tag }}
        {{ form.contrasena }}
        <span id="eyeIcon" class="eye-icon">&#128065;</span>
    </div>

    <button type="submit">Registrarse</button>
</form>
```

Creamos el formulario con tres campos de texto vinculados con los formularios creados en el archivo “forms.py” y que se pasa por parámetro en el archivo “views.py”.

Además, en el campo de la contraseña, creamos el icono del ojo que va a tener la misma configuración y finalidad que el que tenía en el archivo “login.html”.

Por último, creamos el botón registrarse.

Creamos el codigo JavaScript, con la misma estructura y funcionalidad que el creado en el archivo anterior (mostrar y dejar de mostrar la contraseña con la funcionalidad del ojo), entre las etiquetas “script”.

```
<script>
    // Función para mostrar u ocultar la contraseña
    document.getElementById('eyeIcon').addEventListener('click', function() {
        var passwordField = document.querySelector('input[name="contrasena"]');
        var eyeIcon = document.getElementById('eyeIcon');

        if (passwordField.type === "password") {
            passwordField.type = "text";
            eyeIcon.innerHTML = "&#128065;&#10060;"; // Ojo cerrado
        } else {
            passwordField.type = "password";
            eyeIcon.innerHTML = "&#128065;"; // Ojo abierto
        }
    });
</script>
```

INICIO

Seguimos con el archivo “incio.html”, configurado para que sea la página que sea capaz de reconocer los animales y que permita subir ciertos archivos (fotos) de animales para que sean reconocidos.

```
<header>
    <h1 class="titulo">WildDetect</h1>
    <!-- Creación del icono del menú hamburguesa -->
    <div class="menu-hamburguesa" id="hamburger-menu">
        <div></div>
        <div></div>
        <div></div>
    </div>
    <!-- Menú de navegación -->
    <ul class="nav-menu" id="nav-menu">
        <li><a href="#">Inicio</a></li> <!-- Ubicación actual -->
        <li><a href="{% url 'reconocimientos' %}">Últimos reconocimientos</a></li>
        <li><a href="{% url 'ajustes' %}">Ajustes</a></li>
        <li><a href="#" onclick="mostrarMensajeCerrarSesion()"><strong>Cerrar Sesión</strong></a></li>
    </ul>
</header>
```

En la cabecera, incluiremos un título, con un menú hamburguesa. Configuramos primero, el icono del menú hamburguesa (con tres rayas), y el menú desplegable de navegación que aparecerá a la hora de pulsar dicho icono, en el que tendremos 4 opciones:

1. Página de inicio (en la que estamos)
2. Página de reconocimiento (página donde se muestran todos los animales captados por el usuario)
3. Página de ajustes (donde se puede acceder a tus datos personales y realizar acciones como eliminar la cuenta o cambiar la contraseña de esta)
4. Cerrar Sesión (botón que cerrará la sesión del usuario (sin eliminar nada) y que te enviará a la página de inicio de sesión)

```
<div class="container">
    <p>Sube una imagen de un animal o usa tu cámara para tomar una foto.</p>

    <form id="formArchivo" action="{% url 'resultado' %}" method="POST" enctype="multipart/form-data" onsubmit="return validarArchivo()">
        {% csrf_token %}
        <input type="file" accept="image/*" id="inputArchivo" name="archivo">
        <br>
        <button class="btn" type="button" id="abrirCamara">Abrir Cámara</button>
        <button class="btn" type="button" id="realizarFoto" style="display:none;">Realizar Foto</button>
        <button class="btn" type="submit">Reconocer archivo</button>

        <!-- Aquí se guarda la foto capturada para enviarla -->
        <input type="hidden" name="fotoBase64" id="fotoBase64">
    </form>

    <br>
    <video id="video" autoplay style="display:none;"></video>
    <canvas id="canvas" style="display:none; margin-top:20px; max-width:100%;"></canvas>
</div>
```

Creamos un contenedor en el que vamos a tener un texto, y un botón en donde vamos a poder subir archivos de tipo imagen. Además, tendremos un botón que podrá abrir la cámara a la hora de pulsarse, accediendo a la cámara del dispositivo.

La cámara con lo que se está mostrando, se pondrá justo debajo de los dos botones, para que el usuario tenga un fácil acceso a lo que está mostrando la cámara.

A la hora de pulsar el botón de Abrir Cámara, la página web, mostrará un botón de “Realizar Foto”, en el que el usuario cuando lo pulse, se captará una foto y será la que analicen los modelos de IA.

Por último, a la hora de realizar la foto, esta se mostrará justo debajo de la imagen de la cámara en tiempo real.

```
<!-- Formulario de cerrar sesión -->
<div id="cerrarSesion" class="modal">
    <div class="modal-content">
        <p>¿Estás seguro de que quieres cerrar sesión?</p>
        <button onclick="cerrarSesion()">Sí</button>
        <button onclick="ocultarMensaje()">No</button>
    </div>
</div>
```

Creamos el formulario que se va a mostrar a la hora de pulsar el botón de cerrar sesión (para evitar que el usuario cierre sesión por error).

Este contenedor tendrá un texto informativo con dos botones, uno que cerrará la sesión y otro que ocultará el contenedor y no ocurrirá nada, configurado mediante funciones.

Estas funciones, estarán configuradas en lenguaje JavaScript, mediante las etiquetas “script”.

```
// Mostrar mensaje de cerrar sesión
function mostrarMensajeCerrarSesion() {
    document.getElementById("cerrarSesion").style.display = "block";
}
```

Función creada para que se muestre el mensaje de cerrar sesión, que saldrá únicamente cuando se pulse el botón y evitando que se muestre cada vez que se recarga la página.

```
// Redirigir si pulsa "Sí"
function cerrarSesion() {
    window.location.href = "/"; // Redirige al login
}
```

Función creada para cerrar sesión, cuando se pulse el botón “si” en el formulario que se muestra ya explicado.

Esta función, redirige al usuario a la pantalla de login.

```
// codigo para que hacer si pulsa el boton "No"
function ocultarMensaje() {
    document.getElementById("cerrarSesion").style.display = "none";
}
```

Función creada para cuando se pulsa el botón “no”, para que se

oculte el mensaje cuando se pulsa el botón.

```
// código para abrir el menu hamburguesa
document.getElementById("hamburger-menu").addEventListener("click", function() {
    const navMenu = document.getElementById("nav-menu");
    navMenu.classList.toggle("active");
});
```

Configuramos, además, el código para abrir el menú hamburguesa cuando se pulsa el icono, cogiendo el id que tiene el icono del menú hamburguesa, haciendo que cuando se pulse el icono, se ponga en activo, o no, dependiendo si está inicialmente el contenedor abierto, o por el contrario, no se está mostrando y se tiene que abrir.

```
// Funcionalidad de cámara y foto
const video = document.getElementById("video");
const btnAbrir = document.getElementById("abrirCamara");
const btnFoto = document.getElementById("realizarFoto"); // Botón "realizar foto"
const canvas = document.getElementById("canvas");
const inputFotoBase64 = document.getElementById("fotoBase64");
```

Almacenamos el elemento video, el botón de abrir cámara, el botón de realizar foto, el canvas donde se mostrará la foto tomada y la imagen captura en variables. Está ultima, almacenará la imagen en formato base64 en su variable.

```
btnAbrir.addEventListener("click", function () {
    if (navigator.mediaDevices && navigator.mediaDevices.getUserMedia) {
        navigator.mediaDevices.getUserMedia({ video: true })
            .then(function (stream) {
                video.srcObject = stream;
                video.style.display = "block";
                btnFoto.style.display = "inline-block";
            })
            .catch(function (error) {
                console.error("Error al acceder a la cámara: ", error);
                if (error.name === "NotAllowedError") {
                    alert("El acceso a la cámara ha sido denegado. Por favor, revisa los permisos de tu navegador.");
                } else if (error.name === "NotFoundError") {
                    alert("No se encontró ninguna cámara disponible.");
                } else {
                    alert("Error desconocido");
                }
            });
    } else {
        alert("No soporta el acceso a la cámara.");
    }
});
```

Cuando pulsamos el botón de “Abrir Cámara”:

Primero comprobamos que el navegador soporta la API de la cámara. Si es así, se solicita permiso para acceder a la cámara del dispositivo. Si el usuario acepta dicho permiso, ponemos la cámara en formato video (stream) y la mostramos. A su vez, mostramos el botón “Realizar foto” para que el usuario pueda hacer una foto en cualquier momento.

Si el usuario rechaza el permiso de usar la cámara, no se encuentra una cámara en el dispositivo, o surge cualquier otro error, saltará un mensaje de error personalizado al usuario. Además, si el navegador del usuario no soporta la API de la cámara, también mostraremos un mensaje de error al usuario.

```
btnFoto.addEventListener("click", function () { // Al hacer click en "realizar foto"
    const context = canvas.getContext("2d");
    canvas.width = video.videoWidth;
    canvas.height = video.videoHeight;
    context.drawImage(video, 0, 0, canvas.width, canvas.height); // Capturamos la imagen
    canvas.style.display = "block"; // Mostramos la imagen capturada

    // Guardamos la imagen como Base64
    const dataUrl = canvas.toDataURL("image/png");
    inputFotoBase64.value = dataUrl;
    alert("Foto realizada. Puedes enviarla con el botón 'Reconocer archivo'.");
});
```

El canvas, es un “lienzo” que se dibuja sobre la página web, en el que podemos poner cualquier elemento sobre el.

A la hora de pulsar el botón “Realizar Foto”:

Primero, se obtiene el contexto 2D del canvas, para poner la foto capturada sobre él.

Después, ajustamos el tamaño del canvas, al mismo que el del video, ya que la imagen a poner va a ser la misma que la del video y tomamos la foto del video.

Posteriormente, mostramos la imagen capturada y la guardamos como base64, para enviarla al backend y que sea predicha por los modelos.

Por último, mostramos un mensaje de éxito al usuario para saber que la foto ha sido tomada con éxito y que puede ser predicha por los modelos de IA de forma exitosa.

```
// Función que valida si el archivo seleccionado es válido
function validarArchivo() {
    const archivo = document.getElementById("inputArchivo").files[0];
    const fotoBase64 = document.getElementById("fotoBase64").value;

    if (!archivo && !fotoBase64) { // Si no se ha seleccionado ningún archivo ni foto
        alert("Por favor, selecciona un archivo o toma una foto antes de continuar.");
        return false;
    }
    return true;
}
```

La función de validar archivo, que va a ser utilizada para saber si el formato de la foto que se pasa a la hora de pulsar el botón “Reconocer Archivo”, es válido o no. Para ello, obtenemos el archivo que se ha pasado a través del botón “Seleccionar Archivo” y la foto tomada por el usuario en caso de que haya abierto la cámara y tomado una foto en el momento.

Si alguna de las dos variables tiene información, es decir, que no están vacías, devolveremos que el archivo pasado es válido. En caso contrario, saltaremos un mensaje de error para que sepa el usuario que debe de subir una foto o realizarla en el momento para que la aplicación funcione de manera correcta.

HISTORIAL DE RECONOCIMIENTOS

En el archivo de “reconocimiento.html”, es el archivo donde se configura toda la parte visual de la pantalla donde se muestran los animales que han sido reconocidos, la hora y la fecha

exacta en el que el animal ha sido reconocido y, además, la posicion GPS en el que el animal fue encontrado.

El archivo contiene el CSS incorporado.

Dentro de las etiquetas “body” y “header” encontramos lo siguiente:

```
<header>
    <h1 class="titulo">WildDetect</h1>
    <!-- creamos la interfaz del menú hamburguesa con tres rayas -->
    <div class="menu-hamburguesa" id="hamburger-menu">
        <div></div>
        <div></div>
        <div></div>
    </div>
    <!-- creamos las distintas opciones de navegacion del menú hamburguesa -->
    <ul class="nav-menu" id="nav-menu">
        <li><a href="{% url 'inicio' %}">Inicio</a></li>
        <li><a href="#">Últimos reconocimientos</a></li>
        <li><a href="{% url 'ajustes' %}">Ajustes</a></li>
        <li><a href="#" onclick="mostrarMensajeCerrarSesion()"><strong>Cerrar Sesión</strong></a></li>
    </ul>
</header>
```

Incluimos la configuración del menú hamburguesa, anteriormente explicada, y que estará en todos los HTML que haya una vez el usuario haya iniciado sesión en la aplicación.

Una vez salimos fuera de las etiquetas “header”, nos encontramos:

```
<div id="cerrarSesion" class="modal">
    <div class="modal-content">
        <p>¿Estás seguro de que quieres cerrar sesión?</p>
        <button onclick="cerrarSesion()">Sí</button>
        <button onclick="ocultarMensaje()">No</button>
    </div>
</div>
```

Además, incluimos el contenedor que sale a la hora de pulsar en el botón de “Cerrar Sesión” del menú hamburguesa.

También, configuramos lo que aparecerá en el cuerpo de la página:

```
<!-- Filtro de animales -->
<input type="text" id="filtrar" placeholder="Filtrar por animal...">
<button id="btn" onclick="filtrarAnimales()">Filtrar</button>

<!-- Mapa -->
<div id="map"></div>
```

Configuramos un campo de texto y un botón en el que el usuario va a poder escribir en él, con la finalidad de poder filtrar los animales. Esto hará que cuando pulsemos al botón “Filtrar”, aparecerán en el mapa los animales que contengan las letras o los animales que tengan el nombre que ha filtrado el usuario en el campo de texto, desapareciendo todos los demás.

Justo debajo, hacemos que el mapa aparezca debajo del campo de texto y el botón para filtrar los animales. La configuración para que aparezca el mapa en la aplicación, la explicaremos ahora.

Dentro de las etiquetas “script”, es decir, el código JavaScript, aparecerá lo siguiente:

```
let map; //variable que contiene el mapa que se va a mostrar
let markers = []; // array que contiene los marcadores de los animales de la base de datos
```

Primero, inicializamos dos variables, la primera, que va a contener el mapa que se va a mostrar, y otra, que va a almacenar el array que contiene los marcadores de los animales de la base de datos que se van a mostrar (nombre del animal y posición GPS).

```
function initMap() {
    // Crear mapa centrado en Madrid
    map = L.map('map').setView([40.4168, -3.7038], 6);

    // Agregar capa de OpenStreetMap, necesario para poder mostrar correctamente el mapa
    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        attribution: '&copy; OpenStreetMap contributors'
    }).addTo(map);

    //recorremos los animales que se obtienen de la base de datos
    {% for animal in animales %}
        // Usar un nombre único para cada marcador con forloop.counter
        //creamos un marcador con las coordenadas en las que se ha encontrado cada animal
        let marker_{{ forloop.counter }} = L.circleMarker([{{ animal.lugarEncuentro.latitud }}, {{ animal.lugarEncuentro.longitud }}]
            color: 'green',  // El color del borde
            fillColor: 'green',  // El color de fondo del círculo
            fillOpacity: 0.5,  // Opacidad del relleno
            radius: 10  // El radio del círculo
        })
            .addTo(map)
            //añadimos una ventana emergente cuando el raton se posa encima de la ubicacion mostrando esta informacion
            .bindPopup('<h3>{{ animal.nombreAnimal }}</h3><p>Fecha y hora: {{ animal.fechaHora }}</p>');

        // Añadir el marcador al array de marcadores
        markers.push({
            nombre: '{{ animal.nombreAnimal }}',
            marker: marker_{{ forloop.counter }} // Usar el nombre único
        });
    {% endfor %}
}
```

Creamos la función initMap, en la que lo que primero que hacemos es guardar en la variable creada anteriormente “map”, el objeto mapa con el id “map” el cual se centra en Madrid (latitud y longitud), con un nivel de zoom del 6.

Después, agregamos una capa de OpenStreetMap, la cual es necesaria para poder agregar el mapa a la página web. Para acceder al mapa, usamos los servidores públicos de OpenStreetMap, que nos dan acceso al mapa mundial, citando a los autores de los mapas, para que su uso sea legal. Por último, añadimos esta capa de OpenStreetMap al mapa creado anteriormente y almacenado en la variable “map”.

Más tarde, recorremos los animales almacenados en la base de datos, mediante un bucle que recorre la variable “animal”, y creamos el marcador. Este marcador se va a almacenar en una variable con un nombre distinto para cada animal que esté almacenado (“marker1”, “marker2”, “marker3” …). El marcador va a almacenar un marcador en forma de circulo, con unos datos de latitud y longitud determinados dependiendo del animal que se esté recorriendo. Además, este marcador será de color verde, con radio y opacidad de este configurados. Por último, añadimos el marcador al mapa y una ventana emergente cuando el ratón se posa y pincha encima del marcador, que mostrará el nombre del animal y la fecha y la hora a la que ha sido encontrado ese animal en la ubicación del marcador.

También añadimos el marcador a un array de marcadores con la finalidad de poder filtrar los marcadores por nombre. Lo explicamos a continuación.

```
//funcion usada cuando pulsamos el boton de filtrar animales
function filtrarAnimales() {
    // Obtenemos el valor del filtro que ha introducido el usuario
    const animalFiltro = document.getElementById("filtrar").value.toLowerCase();

    // Recorremos los marcadores y mostrar solo los que coincidan con el filtro
    markers.forEach(function(animal) {
        //cogemos el nombre del animal de cada uno de los marcadores (en minuscula)
        const nombreAnimal = animal.nombre.toLowerCase();

        // Mostramos el marcador si el nombre del animal incluye el filtro
        if (nombreAnimal.includes(animalFiltro)) {
            animal.marker.addTo(map);
        } else {
            map.removeLayer(animal.marker); // Eliminar el marcador si no coincide
        }
    });
}
//hacemos que se ejecute la funcion de initMap (mapa con todos los marcadores), cuando se cargue la página por completo
//DOMContentLoaded se asegura de que se cargue el codigo JavaScript cuando el HTML haya cargado correctamente
document.addEventListener('DOMContentLoaded', initMap);
```

Esta función realizará el filtrado de animales cuando el usuario pulse el botón de filtrar.

Primero cogemos el valor de lo que puso el usuario en el campo de texto en el momento en el que este pulsa el botón de filtrar y lo guardamos en minúsculas para evitar fallos innecesarios.

Se recorre el array que contiene los animales (nombre del animal y el marcador o posición en el mapa) y guarda el nombre del animal que está recorriendo en minúsculas.

Si el nombre del animal guardado contiene las letras que haya puesto el usuario en el campo de texto, se añade el marcador (nombre del animal y su posición en el mapa) al mapa. En caso de que el nombre del animal no esté incluido en el filtro, se elimina el marcador del mapa.

```
// Inicializar el mapa cuando se haya cargado la página
document.addEventListener('DOMContentLoaded', initMap);
```

Una vez hecho esto, inicializamos el mapa una vez cargue la página (mediante el evento “DOMContentLoaded”), para que el mapa aparezca de forma automática.

Después, en una nueva etiqueta “script” incluiremos el siguiente código:

```
<script>
    // Funciones de cierre de sesión y menú hamburguesa
    function mostrarMensajeCerrarSesion() {
        document.getElementById("cerrarSesion").style.display = "flex";  // Mostrar modal
    }
    function cerrarSesion() {
        window.location.href = "/"; // Redirige al login
    }
    function ocultarMensaje() {
        document.getElementById("cerrarSesion").style.display = "none";
    }

    // Activar el menú hamburguesa
    document.getElementById("hamburger-menu").addEventListener("click", function() {
        const navMenu = document.getElementById("nav-menu");
        navMenu.classList.toggle("active");
    });

    document.addEventListener('DOMContentLoaded', function() {
        document.getElementById("cerrarSesion").style.display = "none";
    });
</script>
```

En esta etiqueta configuramos en las tres primeras funciones el uso del menú hamburguesa (ya explicado anteriormente), con las funciones realizadas para mostrar el mensaje de iniciar sesión cuando se pulse el botón en el menú hamburguesa, la función que cierra sesión cuando se pulsa “si” en el contenedor anterior, la función que oculta el mensaje cuando se pulsa “no” en el contenedor anterior, la función que activa el contendor y lo desactiva cada vez que el usuario pulsa en el icono, y, la función que mantiene oculto este contenedor para evitar que este aparezca cuando se recargue la página.

AJUSTES

Seguimos con el archivo “ajustes.html”, en el que configuramos la página de ajustes de la aplicación web. En este archivo, integramos el CSS junto con el HTML, en el que también tenemos código JavaScript integrado. Comenzamos a analizar el código:

```
<input type="hidden" name="csrfmiddlewaretoken" value="{{ csrf_token }}">
<header>
    <h1 class="titulo">WildDetect</h1>
    <div class="menu-hamburguesa" id="hamburger-menu">
        <div></div>
        <div></div>
        <div></div>
    </div>
    <!-- Menú de navegación -->
    <ul class="nav-menu" id="nav-menu">
        <li><a href="{% url 'inicio' %}">Inicio</a></li>
        <li><a href="{% url 'reconocimientos' %}">Últimos reconocimientos</a></li>
        <li><a href="#">Ajustes</a></li>
        <li><a href="#" onclick="mostrarMensajeCerrarSesion()"><strong>Cerrar Sesión</a></li>
    </ul>
</header>
```

Al igual que en todas las páginas posteriores a la de iniciar sesión o registrarse, tenemos en las etiquetas “header”, la configuración de la vista del menú hamburguesa, anteriormente explicada. Además, como añadido a otros archivos HTML en este apartado, hemos añadido de manera manual el token “csrf” (obligatorio en todos los métodos POST).

Fuera de la etiqueta “header”, en el cuerpo de la página, encontraremos lo siguiente:

```
<div class="settings-container">
    <h1>Configuración de la cuenta</h1>
    <br>
    <img src="{% static 'img/cuenta.png' %}" alt="Imagen de la cuenta" width="180">
    <br>
    <h3>Bienvenido {{ usuario }}</h3>
    <h5>Correo: {{email}}</h5>
    <br>
    <button onclick="mostrarMensajeCambiarContrasena()">Cambiar Contraseña</button>
    <br>
    <button onclick="mostrarMensajeEliminarCuenta()">Eliminar Cuenta</button>
    <br>
</div>
```

En este apartado creamos un contendor, en el que introduciremos un texto “Configuración de la cuenta” como titulo de página. Justo debajo, añadiremos una imagen con el emoticono típico de una cuenta de usuario, poniendo justo después un mensaje personalizado al usuario con su nombre de usuario, proporcionandole datos de su cuenta como el correo que utilizó en el momento de crear la cuenta y con el que ha iniciado la sesión.

También creamos dos botones con distinta finalidad. El primer botón es utilizado para que el usuario pueda cambiar la contraseña, y el segundo, para que el usuario pueda eliminar la cuenta de la base de datos, perdiendo así, toda la informacion referente a dicho usuario.

```
<div id="cerrarSesion" class="modal">
    <div class="modal-content">
        <p>¿Estás seguro de que quieres cerrar sesión?</p>
        <button onclick="cerrarSesion()">Sí</button>
        <button onclick="ocultarMensaje()">No</button>
    </div>
</div>
```

Creamos el contenedor que nos saldrá a la hora de pulsar el botón de iniciar sesión en el menú hamburguesa, creado en todas las páginas anteriores en las que usábamos el menú.

```
<div id="eliminarCuenta" class="modal">
    <div class="modal-content">
        <h3 style="color: red">¿Estás seguro de que quieres eliminar la cuenta?</h3>
        <p>Se perderá toda la información</p>
        <button onclick="eliminarCuenta1()">Sí</button>
        <button onclick="ocultarMensajeEliminarCuenta()">No</button>
    </div>
</div>
```

Creamos otro contendor, en este caso, es el que aparecerá en el momento en el que el usuario pulsa el botón de eliminar su cuenta de usuario. En este contendor, tenemos un texto en el que se advierte de la acción que se va a realizar, en color rojo, y dos botones, uno para eliminar la cuenta de manera definitiva, y otro, para cancelar la acción. Estos botones están ligados a unas funciones que se generaran a continuación.

```
<div id="cambiarContrasena" class="modal" style="display: none;">
    <div class="modal-content">
        <h2>Cambiar Contraseña</h2>
        <form id="formCambiarContrasena" onsubmit="cambiarContrasena(event)">
            {% csrf_token %}
            <label for="actual">Contraseña Actual:</label>
            <input type="password" id="actual" name="actual" required>
            <br><br>
            <label for="nueva">Nueva Contraseña:</label>
            <input type="password" id="nueva" name="nueva" required>
            <br><br>
            <label for="confirmar">Confirmar Nueva Contraseña:</label>
            <input type="password" id="confirmar" name="confirmar" required>
            <br><br>
            <button type="submit">Guardar Cambios</button>
            <button type="button" onclick="cancelarContrasena()">Cancelar</button>
        </form>
    </div>
</div>
```

En este último contenedor creamos la informacion que va a salir cuando el usuario pulse el botón de cambiar contraseña. En este contendor tendremos un título en el que se mostrará la acción que va a realizar el usuario con varios campos de texto. En el primer campo de texto habrá que poner la contraseña actual del usuario, para que el usuario pueda realizar el cambio de contraseña. Esto se le pide al usuario por cuestiones de seguridad. En el segundo campo de texto el usuario insertará la nueva contraseña que quiere configurar, poniendo lo mismo en el tercer campo, que se ha puesto por motivos de seguridad, con la finalidad de que el usuario no ponga una contraseña de forma equivocada por error.

Por último, ponemos dos botones, el primero que sirve para que se guarden los cambios y se configure la nueva contraseña, y el segundo, para cancelar la acción de cambiar la contraseña.

```
<script>
    // código para abrir el menu hamburguesa
    document.getElementById("hamburger-menu").addEventListener("click", function() {
        const navMenu = document.getElementById("nav-menu");
        navMenu.classList.toggle("active");
    });
    // codigo para mostrar el alert de Cerrar Sesión
    function mostrarMensajeCerrarSesion() {
        document.getElementById("cerrarSesion").style.display = "block";
    }
    // codigo para que hacer si pulsa el boton "Si"
    function cerrarSesion() {
        window.location.href = "/"; // Redirige al login
    }
    // codigo para que hacer si pulsa el boton "No"
    function ocultarMensaje() {
        document.getElementById("cerrarSesion").style.display = "none";
    }
```

Cacho de código de JavaScript que se configura las acciones que va a realizar el menú hamburguesa, explicado anteriormente y configurada en todas las páginas anteriores posteriores al inicio de sesión y registro.

```
//codigo para mostrar el mensaje completo de mostrar contraseña
function mostrarMensajeCambiarContrasena() {
    document.getElementById("cambiarContrasena").style.display = "flex";
}

//boton de cancelar contraseña
function cancelarContrasena() {
    document.getElementById("cambiarContrasena").style.display = "none";
}
```

Funciones realizadas para mostrar el mensaje de cambiar la contraseña cuando el usuario pulse el botón, además de la función configurada para que desaparezca el contenedor emergente cuando el usuario da al botón de cancelar.

```
//funcion para cambiar la contraseña al usuario que ha iniciado sesion
function cambiarContrasena(event) {
    event.preventDefault(); // Evita que el formulario se envíe automáticamente

    //cogemos los valores de las tres contraseñas puestas por el usuario
    let actual = document.getElementById("actual").value;
    let nueva = document.getElementById("nueva").value;
    let confirmar = document.getElementById("confirmar").value;

    //ponemos que todos los campos sean obligatorios de rellenar
    if (!actual || !nueva || !confirmar) {
        alert("Todos los campos son obligatorios");
        return;
    }

    //las contraseñas nuevas deben de coincidir
    if (nueva !== confirmar) {
        alert("Las contraseñas no coinciden");
        return;
    }
```

Pasamos a la funcion creada para cambiar la contraseña del usuario cuando este pulse al botón “Guardar Cambios” del contenedor emergente.

Primero, hacemos que el formulario no se pueda enviar de manera automática, por si surge algún error en el formulario, cancelar el envío del mismo, y evitar problemas para el usuario, con la funcion “preventDefault()”.

Después, guardamos en las distintas variables lo que el usuario ha introducido en los tres campos de texto y comprobamos que todos los campos de texto contengan algo en su interior, sino, saltará un error.

Comprobamos que las dos contraseñas que hemos puesto (la contraseña que el usuario quiere poner nueva comparado con la contraseña nueva de confirmación). En caso de que estas contraseñas no coincidan, se mostrará un error, ya que, es obligatorio que estas dos coincidan.

```
// Validación de la contraseña nueva (mínimo 8 caracteres, una mayúscula, una minúscula y un número)
const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d).{8,}$/;
if (!passwordRegex.test(nueva)) {
    alert("La nueva contraseña debe tener al menos 8 caracteres, una mayúscula, una minúscula y un número.");
    return;
}
```

Validamos que la contraseña, tenga al menos 8 caracteres, una letra minúscula, una letra mayúscula y un número. Si la contraseña no cumple estos parámetros, mostramos un mensaje de error al usuario.

```
    //generamos una peticion HTTP para cambiar la contraseña
    fetch("{% url 'cambiar_contrasena' %}", {
        //la solicitud será de tipo POST
        method: "POST",
        //en el encabezado de la solicitud se indica
        headers: {
            //que el cuerpo de la solicitud será un objeto JSON
            "Content-Type": "application/json",
            //cogemos el token insertado en el formulario (automaticamente) para añadirlo a la cabezera de la solicitud (asegura q
            "X-CSRFToken": document.querySelector("[name=csrfmiddlewaretoken]").value
        },
        // el cuerpo proviene de JSON, que contiene la contraseña actual y la contraseña nueva
        body: JSON.stringify({
            actual: actual,
            nueva: nueva
        })
    })
    //la respuesta de la solicitud se maneja en formato JSON
    .then(response => response.json())
    .then(data => {
        //se controla el tipo de dato de salida. Si es satisfactoria
        if (data.success) {
            alert("Contraseña cambiada con éxito");
            //cambiamos de id del formulario
            document.getElementById("cambiarContrasena").style.display = "none";
            document.getElementById("formCambiarContrasena").reset();
        } else {
            //Si no es satisfactoria se muestra un mensaje de error
            alert("Error: " + data.error);
        }
    })
    //Si sucede algún error durante el proceso, saltará el error
    .catch(error => console.error("Error:", error));
}
```

Seguimos con la función de cambiar la contraseña, en la que enviamos una petición POST para realizar el cambio de contraseña, enviando un archivo JSON con protección contra ataques CSRF en Django para proteger el archivo.

Antes de enviar el archivo y realizar la petición, convierte los valores de la contraseña nueva y la actual a formato JSON.

Una vez enviado, manejamos la respuesta de la petición en formato JSON. El dato de salida es controlado, haciendo que, si este es satisfactorio, se saque un mensaje de éxito, diciendo “la contraseña ha sido cambiada con éxito” y reseteamos el formulario, dejando los campos en blanco y escondiéndole.

En caso de que el formulario no tenga una respuesta JSON exitosa, mostramos un mensaje de error.

Por último, ponemos un catch, que capte los errores que pueda haber no detectados durante la ejecución de la función.

```
//funcion que muestra el formulario para eliminar la cuenta
function mostrarMensajeEliminarCuenta() {
    document.getElementById("eliminarCuenta").style.display = "block";
}
```

Creamos la función que se va a encargar de mostrar el mensaje de eliminar la cuenta (contenedor) cuando se pulsa al botón correspondiente en la página web.

```
//funcion que elimina la cuenta mediante otra peticion HTTP (post)
function eliminarCuenta1() {
    const csrfToken = getCSRFToken(); // Obtener el token CSRF
    if (!csrfToken) {
        console.error("No se encontró el token CSRF");
        return;
    }

    fetch(`/eliminarCuenta/`, {
        method: 'POST',
        headers: {
            'X-CSRFToken': csrfToken, // Incluir el token en las cabeceras
        }
    })
    .then(response => response.json())
    .then(data => {
        //si la respuesta de la peticion es satisfactoria
        if (data.status === 'success') {
            window.location.href = "/"; // Redirigir al login o página de inicio
        } else {
            //Si no es satisfactoria, se muestra el mensaje que se ha obtenido
            alert(data.message);
        }
    })
    //capturamos los errores que pueda haber durante el proceso
    .catch(error => {
        console.error('Error: ', error);
        alert("Hay un error al eliminar la cuenta");
    });
}
```

Creamos la función para eliminar la cuenta del usuario cuando pulsa el botón “si” en el contenedor emergente de eliminar la cuenta.

Primero, obtenemos el token CSRF, si este token es nulo, saltará un error.

Después, creamos una petición con método POST. Convertimos la respuesta del servidor a formato JSON. Si la respuesta del servidor es satisfactoria, se redirigirá al usuario a la pantalla de login, en caso contrario, saltará un error.

Además, añadimos un catch, con la finalidad de captar todos los errores que haya durante la ejecución de la función que no hayan sido captados con anterioridad.

```
//funcion que oculta el formulario de eliminar la cuenta (si pulsas cancelar)
function ocultarMensajeEliminarCuenta() {
    document.getElementById("eliminarCuenta").style.display = "none";
}
```

Función creada para ocultar el mensaje de eliminar la cuenta cuando se pulse al botón “no” del contenedor emergente que salta al pulsar el botón de eliminar cuenta.

```
//funcion que controla el funcionamiento del menu hamburguesa
//obtenemos el icono del menu, y decimos que al hacer click sobre el, se muestre o se oculte el elemento "nav-menu" (el menú con l
document.getElementById("menu-hamburguesa").addEventListener("click", function() {
    const navMenu = document.getElementById("nav-menu");
    navMenu.classList.toggle("active");
});
```

Función creada para activar y desactivar el menú hamburguesa, creada en todas las páginas anteriores posteriores al login.

RESULTADOS

Seguimos con la página de resultados, que será la que al pulsar en el botón de “Reconocer archivo”, navegaremos para que los modelos de IA nos muestren el resultado del reconocimiento.

```
<header>
    WildDetect
</header>
```

Creamos un header, que estará configurado como todos los de las páginas anteriores, con la diferencia de que este no tendrá un menú hamburguesa.

```
<div class="container">
    <h2>Resultado del reconocimiento</h2>
    {% if probabilidad >= 60 %}
        <p>El modelo ha detectado un <b>{{clase}}</b> con una probabilidad de <b>{{probabilidad}}%</b></p>

        <a href="{% url 'inicio' %}" class="btn">Volver a reconocer</a>

        <form id="guardarAnimalForm" method="POST" action="{% url 'guardar_animal' %}" onsubmit="return validarUbicacion()">
            {% csrf_token %}
            <input type="hidden" name="clase" value="{{ clase }}">
            <input type="hidden" name="probabilidad" value="{{ probabilidad }}">
            <input type="hidden" name="latitud" id="latitud">
            <input type="hidden" name="longitud" id="longitud">
            <button type="submit" class="btn">Guardar animal</button>
        </form>
    {% else %}
        <p>El modelo no ha podido reconocer con seguridad el animal de la imagen</p>
        <a href="{% url 'inicio' %}" class="btn">Volver a reconocer</a>
    {% endif %}
</div>
```

Creamos un contenedor, en el que mostraremos el resultado del reconocimiento. Creamos el titulo del contenedor, y lo configuraremos de manera que, si el modelo reconoce algún animal con una probabilidad mayor al 60%, mostraremos un mensaje en el que diga la probabilidad con la que se ha reconocido dicho animal, y el animal que ha sido reconocido. Pondremos dos botones “Volver a reconocer”, y “Guardar Animal”.

Además crearemos un formulario en donde se podrá introducir la latitud y la longitud del usuario cuando se den unas condiciones expresas (el sistema no accede a la ubicación del dispositivo debido a que se ha subido un archivo ya creado, es decir, no se ha realizado la foto al momento en la página web, y a la hora de subir la imagen ya creada, la imagen no tiene en su memoria EXIF la ubicación GPS), con la que el usuario podrá poner la ubicación exacta en el momento en el que realizó la foto, guardando esto en la base de datos.

En caso de que los modelos no reconozcan un animal con una probabilidad de mas del 60%, mostraremos un mensaje al usuario en el que diremos que los modelos no han podido reconocer ningún animal con seguridad, y pondremos el botón “Volver a reconocer” en el que a la hora de pulsar, redirigiremos al usuario a la página de inicio.

```
<!-- Modal de error -->
<div id="modalUbicacion" class="modal_ubicacion">
    <div class="modal-content">
        <h2>Ingresar Ubicación. Si pulsas "Cerrar" se guardará una ubicacion predeterminada de la comunidad.</h2>

        <!-- Formulario para latitud y longitud -->
        <form id="ubicacionForm" onsubmit="return false;"> <!-- Prevenimos el envío por defecto -->
            <label for="latitud">Latitud:</label>
            <input type="text" id="latitud_modal" name="latitud_modal" placeholder="Ingresar latitud" required/><br><br>
            <input type="hidden" name="csrfmiddlewaretoken" value="{{ csrf_token }}">
            <label for="longitud">Longitud:</label>
            <input type="text" id="longitud_modal" name="longitud_modal" placeholder="Ingresar longitud" required/><br><br>

            <!-- Botones -->
            <button type="button" onclick="cerrarModal()">Cerrar</button>
            <button type="button" onclick="guardarUbicacionManual()">Guardar Ubicación</button>
        </form>
    </div>
</div>
```

Definimos un contendor emergente (modal), que aparecerá cuando las condiciones de la ubicación de las imágenes dichas anteriormente se den.

Daremos como título información al usuario de lo que tiene que hacer en el formulario, diciendo que si pulsas el botón de “Cerrar” se guardará en la base de datos el animal reconocido en una ubicación predeterminada.

Configuramos la existencia de dos campos de texto, uno para la latitud y otro para la longitud, protegiéndolo con el token “CSRF”, y configurando dos botones “Cerrar” y “Guardar Ubicación”.

Dentro de las etiquetas script, tendremos varias funciones:

```
document.addEventListener("DOMContentLoaded", function() {
    document.getElementById('modalUbicacion').style.display = 'none';
});
```

Configuramos que el contendor emergente para poner la ubicación manual aparezca cerrado siempre que se recargue la página.

```
// Función de validación
function validarUbicacion() {
    const latitud = document.getElementById('latitud').value;
    const longitud = document.getElementById('longitud').value;

    // Si latitud o longitud son nulos, mostramos el modal y evitamos el envío del formulario
    if (!latitud || !longitud) {
        document.getElementById('modalUbicacion').style.display = 'flex'; // Mostrar modal
        return false; // Evitar el envío del formulario
    }
    return true; // Permitir el envío del formulario
}
```

Función en la que validamos la ubicación, obteniendo la longitud y la latitud configuradas, (dependiendo de la imagen subida, la del dispositivo, o la de la imagen dada por el usuario). Si ambas son nulas, se muestra el formulario creado para meter la ubicación manual.

```
function guardarUbicacionManual() {
    const latitud = parseFloat(document.getElementById('latitud_modal').value);
    const longitud = parseFloat(document.getElementById('longitud_modal').value);

    // Validar que sean números válidos y estén en rango
    if (isNaN(latitud) || latitud < -90 || latitud > 90) {
        alert("La latitud debe estar entre -90 y 90.");
        return;
    }

    if (isNaN(longitud) || longitud < -180 || longitud > 180) {
        alert("La longitud debe estar entre -180 y 180.");
        return;
    }

    // Si todo está correcto, asignamos los valores y cerramos el modal
    document.getElementById('latitud').value = latitud;
    document.getElementById('longitud').value = longitud;

    document.getElementById('modalUbicacion').style.display = 'none';

    document.getElementById('guardarAnimalForm').submit();
}
```

Función para guardar la ubicación manual, en el que almacenamos los valores puestos por el usuario en los campos de texto del formulario, comprobamos que los valores de latitud y longitud sean adecuados (existan).

Si estos valores son válidos, asignamos la latitud y la longitud a los campos que hay en “guardarAnimalForm” y ocultamos el contenedor. Por último, llama de manera automática al formulario “guardarAnimalForm”, haciendo este que se guarde dicha ubicación junto con el animal predicho.

```
function cerrarModal() {
    const latitud = 40.6570;
    const longitud = -4.7000;

    document.getElementById('latitud').value = latitud;
    document.getElementById('longitud').value = longitud;

    document.getElementById('modalUbicacion').style.display = 'none';

    document.getElementById('guardarAnimalForm').submit();
}
```

Si pulsamos el botón “Cerrar” en el contenedor emergente, asignamos esta ubicación al animal, cerramos el contenedor y mandamos dicha información al formulario“guardarAnimalForm”, que se encargará de llamar a la vista correspondiente, la encargada de almacenar en la base de datos dicho lugar y animal.

CAMBIAR CONTRASEÑA

```
<header>
    Recuperar Contraseña
</header>
```

Ponemos como título de cabecera “Recuperar contraseña”, sin poner un menú hamburguesa, al igual que en el archivo anterior.

```
<div class="container">
    <h2>Introduce una nueva contraseña</h2>
    {% if messages %}
        <div class="error-messages">
            {% for message in messages %}
                <div class="error-message">{{ message }}</div>
            {% endfor %}
        </div>
    {% endif %}
    <br>

    <form method="POST" action="{% url 'reestablecer_contrasena' %}">
        {% csrf_token %}
        <label for="password1">Nueva contraseña:</label>
        <input type="password" id="password1" name="password1" required>

        <label for="password2">Confirmar nueva contraseña:</label>
        <input type="password" id="password2" name="password2" required>

        <button type="submit" class="btn">Establecer contraseña</button>
    </form>
</div>
```

Creamos un contenedor en el que el usuario podrá cambiar su contraseña.

Ponemos el titulo al contenedor y justo debajo la gestión de errores mediante mensaje.

Seguimos con la creación del formulario, que llamará a la vista encargada de cambiar la contraseña del usuario en la base de datos. En este formulario, creamos dos campos de texto, en los que el usuario deberá de introducir la contraseña nueva, y un botón “Establecer contraseña”, que como configuramos en el apartado “views.py”, a la hora de

pulsarlo se nos cambiará la contraseña en la base de datos y nos redirigirá al “login” para que el usuario pueda iniciar sesión en la página web, o por el contrario, mostrará un error en esta misma pantalla.

ARCHIVOS CSS

La configuración de los CSS no ha sido explicada, ya que por la cantidad de nombres distintos configurados para cada botón, campo de texto, contenedor o cabecera de cada HTML, será complicado entender y explicar mediante fotos para que sirve cada cosa. Accediendo al código, se podrá comprender mejor como está configurado los CSS de cada HTML (teniendo las 3 páginas principales de la página web, inicio, reconocimientos y ajustes un CSS externo conjunto, además del suyo propio).