Paquete frweb

Todas estos métodos se encuentran dentro de frweb.admin_views. Todos ellos son utilizados para el funcionamiento del cliente tipo administrador.

ZonaAdmin:

Vista de inicio del servicio de administración, para acceder y utilizar todos los servicios de esta zona.

Método: **función** ZonaAdmin(request).

Request GET: Vista de inicio del servicio de administración.

context msg: Mensaje de texto para mostrar en la vista.

tipo context: texto.

return.GET: Vista con el mensaje introducidos en el context.

tipo return: HTTP.

Vista: 'templates/Administracion.html'.

```
@staff_member_required
def ZonaAdmin(request):
    context={'msg':'Bienvenido a la zona de Administración'}
    return render(request,'Administracion.html',context)
```

SubirZip:

Recibe y aloja el archivo ".zip" enviado desde los archivos locales del cliente con los datos de la nueva persona a añadir y poder almacenarla en el propio servicio dentro del directorio "/FaceRecognition". El envio se realiza tras cumplimentar el formulario el formulario requerido. Recibido el archivo lo descomprime y aloja las fotos de la nueva persona en el directorio "/FaceRecognition/train_img"

Método: **función** SubirZip(request).

Request GET: Envio de formulario mostrando los datos necesarios para procesar la petición POST.

context form: Formulario a cumplimentar por el cliente para proceder al ingreso de una nueva persona.

formulario: frweb.forms.UploadFileForm`.

tipo context: texto.

return.GET: Vista con el formulario.

tipo return: HTTP.

Vista GET: 'templates/nuevapersona.html'.

Request POST: Subida y alojamiento de la imagen para su analisis.

parámetro Archivo: Archivo local del usuario con las imagenes de la nueva persona a añadir.

tipo parámetro: Archivo .zip.

parámetro Nombre: Nombre con el que se va a desbrir a la nueva persona.

tipo Nombre: texto.

return.POST: HttpResponse devuelve la vista con el estado del proceso.

tipo return: HTTP.

Vista POST: 'templates/respuestasadmin.html'.

```
@staff_member_required
def SubirZip(request):
    if request.method == 'POST':
        form = UploadFileForm(request.POST, request.FILES)
        if form.is_valid():
            # subir archivo
            Archivo = request.FILES['Archivo']
            nombre = request.POST['Nombre']
            #crear rutas de almacenamiento si no existen
            if not os.path.isdir(basedir+ '\\FaceRecognition\\train_img'):
                os.mkdir(basedir+ '\\FaceRecognition\\train_img')
                os.mkdir(basedir+ '\\FaceRecognition\\pre_img')
            # comprobar ruta
            if not os.path.isdir(basedir+ '\\FaceRecognition\\train_img\\' + nombre):
                os.mkdir(basedir+ '\\FaceRecognition\\train_img\\' + nombre)
            fs = FileSystemStorage()
            fs.save(Archivo.name, Archivo)
            # UnzipArchivo
            reg = Usuarios(nombre=nombre)
            reg.save()
            with zipfile.ZipFile((basedir + '\\media\\' + Archivo.name), 'r') as zip_ref:
                print('Extraer')
                zip_ref.extractall(basedir + '\\FaceRecognition\\train_img\\' + nombre)  # modificar ruta destino
            ##Borrar ZIP
            os.remove(basedir + '\\media\\' + Archivo.name)
            context={'estado':'La persona ha sido actualizada correctamente'}
            return render(request,'respuestasadmin.html',context)
    else:
        form = UploadFileForm()
    documents = Persona.objects.all()
    # Render list page with the documents and the form
    context = {'documents': documents, 'form': form}
    return render(request, 'nuevapersona.html', context)
```

Entrenamiento:

Con este método se realiza el entrenamiento obtienen primero los datos almacenados en el directorio '/FaceRecognition/train_img', estos datos se normalizan y se almacenan en el directorio '/FaceRecognition/pre_img', estos son los datos utilizados para realizar el entrenamiento. Hace uso de los metodos alojados en ´FaceRecognition.data_preprocess´ y´FaceRecognition.train_main´.

Método: **función** Entrenamiento(request).

Request GET: Envio del estado del proceso una vez realizado el entrenamiento, si el estado es satisfactorio devuelve la lista de las personas con las que se ha entrado el sistema y el número de las fotos utilizadas, si el estado es no satisfactorio responde con un mensaje negativo.

context registros: Lista de personas reconocidas y número de fotos.

context resultado: Indica el estado de haber aplicado el entrenamiento en el servicio.

tipo context: texto.

return.GET: Vista con el resultado del entrenamiento.

tipo return: HTTP.

Vista: 'templates/Entrenamiento.html'.

```
@staff_member_required
def Entrenamiento(request):
    try:
    #borrar bounding_box para resetear los entrenamientos
    bb = os.listdir(basedir+ '\\FaceRecognition\\pre_img')
    for box in bb:
        b=box.split('_')
        if len(b)>2:
            os.remove(basedir+'\\FaceRecognition\\pre_img\\'+box)
    #borrar lista de registros
    Reconocidos.objects.all().delete()
    Usuarios.objects.all().delete()

    #proceso de entrenamiento
    print("INFO  [Realizando el modelo]")
    entrenar.DPreprocesses.preproceso(None)
    print("INFO [Entrenando el modelo]")
    modelo.Train_main.train_main(None)
    print('Entrenamiento terminado')

    print('Actualizando la base de datos') #actualizar lista
    lista = os.listdir(basedir+ '\\FaceRecognition\\train_img')
    for nombre in lista:
        fotos = os.listdir(basedir+ '\\FaceRecognition\\train_img\\' + nombre)
        p = Reconocidos(nombre=nombre, fotos=len(fotos))
        p.save()
        r=Usuarios(nombre=nombre)
        r.save()
    listado = Reconocidos.objects.all()

    context = {'registros': listado, 'resultado': 'El entrenamiento se ha relizado correctamente'}
    return render(request, 'Entrenamiento.html', context)
except:
    context = {'registros': '' , 'resultado': 'El entrenamiento no se ha podido realizar'}
    return render(request, 'Entrenamiento.html', context)
```

BorrarReconocido:

Con este método se puede borrar todos los datos una persona tipo usuario y reconocido si existiese. El envio se realiza tras cumplimentar el formulario el formulario requerido.

Método: **función** BorrarReconocido(request).

Request GET: Envio de formulario mostrando los datos necesarios para procesar la petición POST.

context form: Formulario a cumplimentar por el cliente para proceder al borrado de una persona.

formulario: *frweb.forms.BorrarName*.

context registros: Lista de personas tipo usuario.

tipo context: texto.

return.GET: Vista con el formulario.

tipo return: HTTP.

Vista GET: 'templates/nuevapersona.html'.

Request POST: Recepción del nombre de la persona a borrar.

parámetro Nombre: Nombre de la persona que se desea borrar.

tipo Nombre: texto.

return.POST: Devuelve los mismos datos que el proceso "Request GET"

tipo return: HTTP.

Vista POST: 'templates/BorrarReconocido.html'.

```
@staff_member_required
def BorrarReconocido(request):
    registros = Usuarios.objects.all()
    lista=''
    for registro in registros:
        lista+=registro.nombre

    if request.method=='POST':
        form=BorrarName(request.POST)
        if form.is_valid():
            #borrar
            nombre=request.POST['Nombre']
            if nombre in lista:
                try:
                    Reconocidos.objects.filter(nombre=nombre).delete()#borraregistro
                    Usuarios.objects.filter(nombre=nombre).delete()#borrar registro
                    if os.path.exists(basedir+ '\\FaceRecognition\\train_img\\' + nombre):#comprueba que existe
                        shutil.rmtree(basedir+ '\\FaceRecognition\\train_img\\' + nombre)#borra carpeta
                    if os.path.exists(basedir+ '\\FaceRecognition\\pre_img\\' + nombre):
                        shutil.rmtree(basedir+ '\\FaceRecognition\\pre_img\\' + nombre)
                    context = {'registros': registros, 'form': form}
                    return render(request, 'BorrarReconocido.html', context)
                except:
                    context = {'registros': registros, 'form': form}
                    return render(request, 'BorrarReconocido.html', context)
            else:
                context = {'registros': registros, 'form': form}
                return render(request, 'BorrarReconocido.html', context)
    else:
        form=BorrarName()
    context = {'registros': registros, 'form': form, 'msg':'Introduce el nombre de la persona que desea borrar'}
    return render(request, 'BorrarReconocido.html', context)
```

mostrarfotoborrado:

Con este método se consiguen y muestran todas las imagenes de una persona tipo usuario. El envio se realiza tras cumplimentar el formulario el formulario requerido.

Método: **función** mostrarfotoborrado(request).

Request GET: Envio de formulario mostrando los datos necesarios para procesar la petición POST.

context form: Formulario a cumplimentar por el cliente para proceder a mostrar la vista de borrado de imagen.

formulario: *frweb.forms.BorrarName*.

context registros: Lista de personas tipo usuario.

context msg: Mensaje de información para el cliente.

tipo context: texto.

return.GET: Vista con el formulario.

tipo return: HTTP.

Vista GET: 'templates/BorrarReconocido.html'.

Request POST: Recepción del nombre de la persona a borrar.

context nombre: Nombre de la persona que se desea borrar o mensaje de resultado negativo en la busqueda de imágenes.

context lista: Listado de la ruta de las imagenes a mostrar si se encuentran y vacia si no se tienen registros.

tipo context: texto.

return.POST: Devuelve la vista de presentación con las imagenes listas para ser borradas

tipo return: HTTP.

Vista POST: 'templates/fotosborrado.html'.

```
@csrf_exempt
@staff_member_required
def mostrarfotoborrado(request):
    registros = Usuarios.objects.all()
    form=BorrarName()
    if request.method=='POST':
        form=BorrarName(request.POST)
        if form.is_valid():
            nombre=request.POST['Nombre']
            if os.path.exists(basedir+'/FaceRecognition/train_img/'+nombre):
                ruta=basedir+'/FaceRecognition/train_img/'+nombre
                listado = os.listdir(ruta)
                context={'lista':listado,'nombre':nombre }
            else:
                context={'lista':"",'nombre':"No se ha encontrado la persona introducida" }
            return render(request, 'fotosborrado.html',context)
    context = {'registros': registros, 'form': form, 'msg':'Introduce el nombre de la persona que desea borrar su foto'}
    return render(request, 'BorrarReconocido.html', context)
```

borrarfoto:

Con este método se obtienen los datos de la foto que se quiere borrar. El envio se realiza tras cumplimentar el formulario el formulario requerido en el método función *frweb.admin_views.mostrarfotoborrado*.

Método: **función** borrarfoto(request).

Request POST: Contiene la cadena de texto QueryDict sin parsear, una vez parseada se obtienen los parámetros necesarios para procesar el borrado.

parámetro nombre: Nombre de la persona que se desea borrar su foto.

tipo nombre: texto.

parámetro foto: Nombre de la foto a borrar

tipo foto: texto

context estado: Listado de la ruta de las imagenes a mostrar si se encuentran y vacia si no se tienen registros.

tipo context: texto.

return.POST: Devuelve la vista de presentación con el estado resuelto de este proceso

tipo return: HTTP.

Vista POST: 'templates/respuestasadmin.html'.

```
@csrf_exempt
@staff_member_required
def borrarfoto(request):
    if request.method=='POST':
        cadt=request.POST.urlencode().split('=&')
        for t in cadt:
            f=t.split('.x')
            if len(f)>1:
                foto=f[0]
        nombre=cadt[2].replace('+'," ").replace('=',"")
        estado='No se ha realizado el borrado'
        try:
            if os.path.exists(basedir+ '\\FaceRecognition\\train_img\\' + nombre +"\\"+ foto):#comprueba que existe
                os.remove(basedir+ '\\FaceRecognition\\train_img\\' + nombre +"\\"+ foto)
            arch=foto.split('.')
            if os.path.exists(basedir+ '\\FaceRecognition\\pre_img\\' + nombre +"\\"+ arch[0]+'.png'):
                os.remove(basedir+ '\\FaceRecognition\\pre_img\\' + nombre +"\\"+arch[0]+'.png')
            estado='El borrado se ha realizado correctamente'
        except:
            print('error en borrado')

        context={'estado':estado}
        return render(request,'respuestasadmin.html',context)
```

---

mostrarfoto:

Con este método muestran todas las fotos de una persona tipo usuario almacenadas en el directorio '/FaceRecognition/train_img'. El envio se realiza tras cumplimentar el formulario el formulario requerido.

Método: **función** mostrarfoto(request).

Request GET: Envio de formulario mostrando los datos necesarios para procesar la petición POST.

- **context form:** Formulario a cumplimentar por el cliente para proceder a mostrar las fotos.
- **formulario:** *frweb.forms.BorrarName*.
- **context registros:** Lista de personas tipo usuario.
- **context msg:** Mensaje para mostrar al cliente.
- **tipo context:** texto.
- **return.GET:** Vista con el formulario.
- **tipo return:** HTTP.

Vista GET: 'templates/BorrarReconocido.html'.

Request POST: Recepción del nombre de la persona a borrar.

- **parámetro Nombre:** Nombre de la persona que se desea ver sus fotos.
- **tipo Nombre:** texto.
- **context lista:** Listado con el nombre de las fotos de la persona deseada.
- **context nombre:** Informa si se ha encontrado la persona devolviendo el nombre de esta o un mensaje negativo.
- **tipo context:** texto.
- **return.POST:** Devuelve los datos procesados para interpretarlos dentro de la vista Post.
- **tipo return:** HTTP.

Vista POST: 'templates/fotosmostrar.html'.

```
@staff_member_required
def mostrarfoto(request):
    registros = Usuarios.objects.all()
    form=BorrarName()
    if request.method=='POST':
        form=BorrarName(request.POST)
        if form.is_valid():
            nombre=request.POST['Nombre']
            if os.path.exists(basedir+'/FaceRecognition/pre_img/'+nombre):
                ruta=basedir+'\\FaceRecognition\\pre_img\\'+nombre
                listado = os.listdir(ruta)
                context={'lista':listado,'nombre':nombre }
            else:
                context={'lista':"",'nombre':"No se ha encontrado la persona introducida" }
            return render(request, 'fotosmostrar.html',context)
    context = {'registros': registros, 'form': form, 'msg':'Introduce el nombre de la persona que desea ver sus fotos de entrenamiento'}
    return render(request, 'BorrarReconocido.html', context)
```

Nota

Todos estos servicios necesitan el usuario y clave de administrador.