Paquete frweb

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

Home:

Vista de inicio del servicio web, para poder acceder y utilizar todos los servicios.

Método: **función** Home(request).

Request GET: Vista de inicio del servicio web.

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/home.html'.

```
def Home(request):
    context={'msg':'Bienvenidos al servicio Web de reconocimiento facial'}
    return render(request, 'home.html', context)
```

ImagenView:

Recibe y aloja una imagen enviada desde los archivos locales del cliente en el propio servicio dentro del directorio '/media'. El envio se realiza tras cumplimentar el formulario el formulario requerido. Analiza la imagen recibida a traves de *FaceRegonition.identify_face_image* para mostrar su resultado al cliente.

Método: **clase** ImagenView(TemplateView).

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 la subida de la imagen.

form referencia: *frweb.forms.ImagenForm*.

tipo context: texto.

return.GET: Vista con el formulario.

tipo return: HTTP.

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

parámetro imagen: Imagen local del usuario.

tipo parámetro: Archivo de imagen.

return.POST: HttpResponseRedirect: Redirecciona el servicio con la respuesta de la imagen analizada

tipo return: HttpResponseRedirect(img).

Vista: 'templates/imagenview.html'.

```
class ImagenView(TemplateView):
    form = ImagenForm
    template_name = 'imagenview.html'
    def post(self, request, *args, **kwargs):

        form = ImagenForm(request.POST, request.FILES)

        if form.is_valid():
            obj = form.save()
            ruta=basedir+"/"+obj.imagen.url
            valor=ifm.identify_face_image.resultado(None, ruta)#devuelva ruta de la nueva foto
            return HttpResponseRedirect(reverse_lazy('Imagendisplay', kwargs={'pk': obj.id}))

        context = self.get_context_data(form=form)
        return self.render_to_response(context)

    def get(self, request, *args, **kwargs):
        #Cada vez que hacemos una petición al servidor borramos las fotos en su interior.
        ruta=(basedir+"/media/images")
        fotos = os.listdir(ruta)
        for foto in fotos:
            os.remove(basedir+"/media/images/"+foto)
        return self.post(request, *args, **kwargs)
```

ImagenDisplay:

Vista de presentación de imagenes analizadas por el servicio de reconocimiento facial.

Método: **clase** ImagenDisplay(DetailView).

Request GET: Vista con los datos de acceso a la imagen analizada por el método "ImagenView".

parametro img: Ruta de la imagen analizada.

tipo parámetro: Archivo de imagen.

return.GET: Vista de representación.

tipo return: HTTP.

Vista: 'templates/imagendisplay.html'.

```
class ImagenDisplay(DetailView):
    model = SubirImagen
    template_name = 'imagendisplay.html'
    context_object_name = 'img'
```

AnalizarVideoVista:

Vista de representación del formulario para realizar el envío del archivo de video necesario para el método "AnalizarVideoFrame".

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

Request GET: Vista para utilizar la plantilla del método "AnalizarVideoFrame" como un elemento iframe de HTML.

return.GET: Vista de representación.

tipo return: HTTP.

Vista: 'templates/AnalizarVideoVista.html'.

```
def AnalizarVideovista(request):
    return render(request, 'AnalizarVideoVista.html')
```

AnalizarVideoFrame:

Recibe y aloja un video enviado desde los archivos locales del cliente en el propio servicio dentro del directorio '/media/videos'.

Analiza los frame del video recibido a traves de *FaceRegonition.identify_face_videoweb* y envia estos al cliente tras cumplimenter el formulario requerido.

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

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

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

context form: Formulario a cumplimentar por el cliente para proceder a la subida del archivo de video.

form referencia: *frweb.forms.UploadVideoForm*

tipo context: texto.

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

tipo return: HTTP.

Request POST: Subida y alojamiento del video para su analisis.

parámetro Video: Video local del usuario.

tipo parámetro: Archivo de video.

return.POST Stream: Devuelve frame a frame la respuesta del analisis del video.

tipo return: StreamingHttpResponse(frame).

Vista: 'templates/AnalizarVideoframe.html'.

```
@xframe_options_sameorigin
@csrf_exempt
def AnalizarVideoframe(request):
    if request.method == 'POST':
        form = UploadVideoForm(request.POST, request.FILES)
        if form.is_valid():
            Archivo = request.FILES['Video']
            fs = FileSystemStorage(basedir+"/media/videos")
            fs.save(Archivo.name, Archivo)
            rArchivo= basedir+"/media/videos"+'/'+Archivo.name
            frame = (ifv.resultado(None, rArchivo))
            return StreamingHttpResponse(frame, content_type='multipart/x-mixed-replace; boundary=frame')
    else:
        form = UploadVideoForm()
        try:
            archivos = os.listdir(basedir+"/media/videos")
            for archivo in archivos:
                os.remove(basedir+"/media/videos/"+ archivo)
        except:
            print('no se ha podido borrar')
    msg='Introduzca el video a analizar'
    context = {'msg':msg, 'form': form}
    return render(request, 'AnalizarVideoframe.html', context)
```

Lista:

Vista de presentación de la lista de todas las personas reconocidas del sistema.

Método: **función** Lista(request).

Request GET: Vista para utilizar la plantilla del método "IpVideoframe" como un elemento iframe de HTML.

context registros: Obtiene todos los valores de la lista de usarios

tipo context: texto

return.GET: Vista de representación.

tipo return: HTTP.

Vista: 'templates/ListaReconocidos.html'.

```
def Lista(request):
    registros=Reconocidos.objects.all()
    context={'registros':registros}
    return render(request,'ListaReconocidos.html',context)
```

IpVideovista:

Vista de representación del formulario para realizar el envío de la dirección Ip necesaria para utilizar método "IpVideoframe".

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

Request GET: Vista para utilizar la plantilla del metodo "IpVideoframe" como un elemento iframe de HTML.

return.GET: Vista de representación.

tipo return: HTTP.

Vista: 'templates/AnalizarVideoVista.html'.

```
@xframe_options_sameorigin
def IpVideovista(request):
    return render(request,'AnalizarVideoIPvista.html')
```

IpVideoframe:

Recibe la dirección Ip que maneja la camara a la que se desea acceder para mostrar el resultado del analisis de su video

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

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

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

context form: Formulario a cumplimentar por el cliente para obtener la dirección Ip.

form referencia: *frweb.forms.VideoForm*

tipo context: texto.

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

tipo return: HTTP.

Request POST: Subida y alojamiento del video para su analisis.

parámetro Ip: Dirección Ip de la camara que se desea visualizar.

tipo parámetro: texto.

return.POST Stream: Devuelve frame a frame la respuesta del analisis del video.

tipo return: StreamingHttpResponse(frame).

Vista: 'templates/AnalizarVideoframe.html'.

```
@xframe_options_sameorigin
@csrf_exempt
def IpVideoframe(request):
    form = VideoForm()
    msg='Introduzca dirección Ip de su camara IP'
    if request.method == 'POST':
        dirIP=request.POST['Ip']
        frame = (ifv.resultado(None, dirIP))
        return StreamingHttpResponse(frame, content_type='multipart/x-mixed-replace; boundary=frame')
    else:
        context={'msg':msg,'form':form}
        return render(request,'AnalizarVideoframe.html',context)
```

---

camvista:

Vista de representación para utiliar la Webcam del cliente y el método postjsimagen de analisis de imagen.

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

Request GET: Vista para utilizar la Webcam del cliente y el método postjsimagen.

return.GET: Vista de representación.

tipo return: HTTP.

Vista: 'templates/webcamjs.html'.

```
def camvista(request):
    return render(request, 'webcamjs.html')
```

---

postjsimagen:

Recibe la imagen capturada a través de la Webcam del cliente, esta imagen se envia en formato texto codificado en Base64 y el servidor descodifica esta imagen y la analiza a través de *FaceRegonition.identify_face_image* , para finalmente responder con la ruta de la imagen analizada por el servidor.

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

Request POST: Envio de la imagen en texto Base64 y el token identificador de la imagen.

parámetro img: Texto en Base64 correspondiente a la captura de imagen realizada por el cliente.

tipo parámetro: texto.

parámetro token: Valor identificativo de la imagen y marca de tiempo de la captura.

tipo parámetro: texto.

return.POST foto: Devuelve el valor de la ruta de la foto analizada.

tipo return: texto.

Vista: 'templates/webcamjs.html'.

```
@csrf_exempt
def postjsimagen(request):
    if request.method == 'POST':
        base=request.POST['img']
        token=request.POST['token']
        time=token.split('token') #se separa el timestamp del identificador
        timeb=int(time[0])-8000000 #se define un tiempo de borrado

        im = Image.open(BytesIO(base64.b64decode(base)))
        foto='media/imagescam/'+token
        im.save(foto, 'PNG')
        fotos = os.listdir(basedir+"/media/imagescam")
        #borrado de fotos mas de 8000000 miliseg
        for archivo in fotos:
            borrado=archivo.split('token')
            if int(borrado[0])<int(timeb):
                os.remove(basedir+"/media/imagescam/"+archivo)

        valor=ifm.identify_face_image.resultado(None, foto)#devuelva ruta de la nueva foto
        return HttpResponse(foto)
```
