# Manual de Usuario
# Servidor Telemetría

Por: Alejandro Corredor
Director: Héctor Cadavid Rengifo

# Tabla de Contenido

# Descripción de la Aplicación

El API REST a disposición permite ingresar tramas de datos desde distintos dispositivos y obtenerlas tras pasar las mismas, por varios filtros digitales. Dándole la libertad a quien lo use, de utilizar las tramas y/o agregar nuevos filtros para obtener los datos deseados en otra aplicación cliente que utilice este API.

# Dirigido A

Este manual está dirigido a futuros usuarios desarrolladores que pretendan usar o extender el sistema actual, ya sea como objeto de proyectos de grado venideros o para usar el conocimiento aquí almacenado, comprendiendo el modelo de actores o el lenguaje Scala entre otras cosas.

# Instalación y puesta en marcha

Como primer paso debe instalarse el manejador de paquetes para el lenguaje de programación Scala. Desde el terminal de Linux ejecutar los siguientes comandos en orden:

```
echo "deb http://dl.bintray.com/sbt/debian /" | sudo tee -a /etc/apt/sources.list.d/sbt.list
sudo apt-get update
sudo apt-get install sbt
```

Posterior a esto se recomienda el IDE Intellij para trabajar con la plataforma. Sin embargo se puede usar el IDE de preferencia que mantenga algún plugin de Scala. Para instalar Intellij basta con dirigirse a su página principal y descargar la aplicación (https://www.jetbrains.com/idea/). Para ejecutarla basta con dirigirse al directorio donde se descargó y en la carpeta bin ejecutar el siguiente comando en la terminal, en el caso de una plataforma Unix:

```
sh ./idea.sh
```

Para ejecutar el proyecto como tal debe de dirigirse a la carpeta “Código Fuente” en el CD de instalación, allí encontrara otro fólder con el nombre “Local”. Tras dirigirse a esta ubicación en la terminal, ejecute el siguiente comando:

```
sbt run
```

Este folder contiene la aplicación principal, la cual procederemos a configurar para bien usar un actor de procesamiento local o uno remoto.

**Instalación de procesamiento Local**

Dentro de la carpeta “Local” acceder a la ruta “app/controllers/webDtos” en la cual encontrara el archivo .scala “ Application”. El cual se encarga de recibir las peticiones HTTP y procesarlas. Para garantizar el procesamiento con el actor local diríjase a la línea  número 104 en el método “getProcActor” se encuentre el siguiente contenido:

```
val actor: ActorRef = system.actorOf(Props (new ActorProc()))
```

Lo anterior, solicita al Sistema de actores de Akka un nuevo actor procesador de carácter local.

**Instalación de procesamiento remoto**

Para poner en funcionamiento el proceso remoto debe realizarse dos configuraciones distintas. Asumiendo falta de conocimiento en este apartado por parte del lector, mantendremos al actor de procesamiento remoto corriendo en el “localHost” con el puerto “5051” en la maquina donde se encuentre instalada el mismo.

La primera parte de la configuración debe hacerse en la aplicación local ubicada en la ruta del CD “Código fuente/Local/app/controllersWebDtos”, en el archivo Application.scala asegúrese de quitar el comentario la siguiente línea de código en el método “getActorProc” sobre la línea 104 del archivo:

```
val actor: ActorRef =

system.actorFor("akka.tcp://HelloRemoteSystem@127.0.0.1:5051/user/RemoteActor")
```

y poner en comentarios la siguiente línea:

```
val actor: ActorRef = system.actorOf(Props (new ActorProc()))
```

Adicionalmente en la primera línea mostrada, después del símbolo “@” se debe cambiar la dirección IP, por la dirección donde se encuentra corriendo la aplicación remota.

Finalmente para poder activar el procesamiento remoto se debe copiar la carpeta “Remoto” contenida en el directorio “Código fuente” del CD, a la maquina donde se desea ejecutar el procesamiento. Luego de elegir el directorio donde se copiara dicha carpeta, debe  navegarse hasta ella en la terminal y ejecutar el comando

```
sbt run
```

# Requerimientos

## 1. Software

Linux Ubuntu 14.04 en adelante
Intellij IDE o Eclipse con Scala Plugin
SBT 13.5
AKKA toolkit
Node.js

## 2. Hardware

Requerimientos mínimos

- Intel Core 2 Duo
- Memoria RAM 1GB
- 500 MB de espacio libre en disco

## 3. Técnicos

- Conocimiento en Java

## 4. Requerimientos de aplicativos web

Para poner en funcionamiento cualquier aplicativo web que desee usar el API que presenta la aplicación, se debe instalar un proxy inverso, con el fin de evitar el problema de correr dos aplicaciones que se comunican entre sí, en distintos dominios. Para lo cual se utilizara la aplicación HAProxy.

Para instalar dicha herramienta basta con ejecutar el siguiente comando en la consola

```
sudo apt-get install haproxy
```

Posteriormente, ha de habilitarse el inicio de esta aplicación como un servicio de Linux, para cual debe ejecutarse el siguiente comando en la terminal:

```
sudo nano/etc/default/haproxy
```

Para finalizar la instalación basta con cambiar el atributo ENABLED a uno en la pantalla de la aplicación nano previamente abierta y guardar el archivo.

### 4.1 Configuración de la aplicativo HAProxy

Para configurar inicialmente el aplicativo, basta con configurar el aplicativo web en el puerto 8000 y sobrescribir el archivo en la ruta /etc/haproxy/haproyx.cfg  por el archivo con el mismo nombre, el cual se encuentra en la carpeta “Codigo Fuente” del presente CD.

Si la aplicación web o el API funcionan en otras direcciones o puertos solo basta con ingresar al archivo y cambiar dichos valores donde estaban los previos, siendo sencillo al ser un archivo de texto muy pequeño e intuitivo

```
global
    maxconn 4096

defaults
    log global
    log 127.0.0.1 local0
    log 127.0.0.1 local1 notice
    mode http
    timeout connect 300000
    timeout client 300000
    timeout server 300000
    maxconn 2000
    option redispatch
    retries 3
    option httpclose
    option httplog
    option forwardfor
#    option httpchk HEAD /check.txt HTTP/1.0

frontend pgr2-front
        bind *:81
        acl tramas url_beg /trama
        use_backend pgr2-back if tramas
        default_backend web-front

backend pgr2-back
    mode http
    server osiris1 0.0.0.0:9000 check

backend web-front
    mode http
    server ang1 0.0.0.0:8000 check
```

# Adicionar un nuevo filtro

Para adicionar un Nuevo filtro al sistema es necesario definir una función, que permita hacer el filtrado. Para lo cual es indispensable definir este ítem en la clase ActorProc, la cual se encuentra en Local/app/models en la carpeta “Código fuente” del CD, quien se encarga de dicho procesamiento.

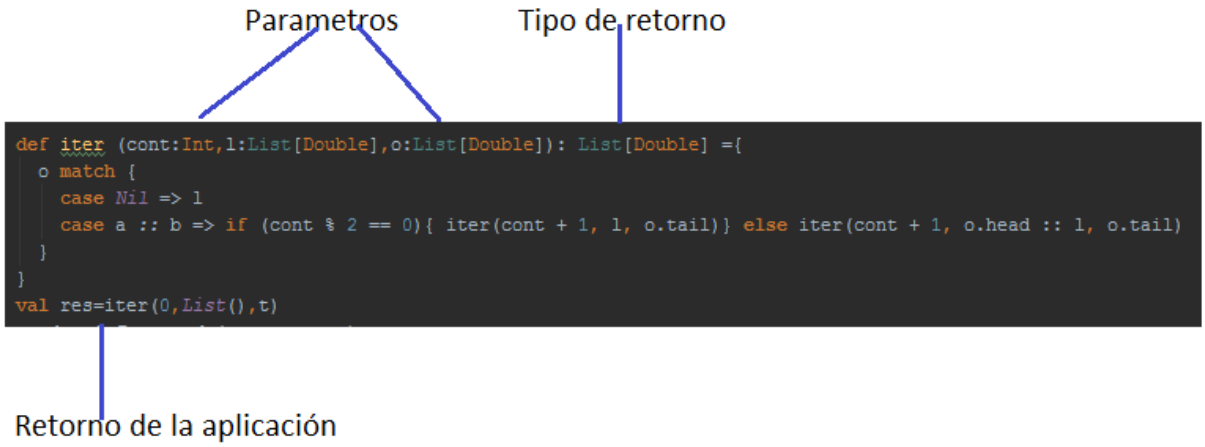
Cabe destacar que la función que recibe los mensajes para procesar trama recibe como parámetros un arreglo de números en formato “doublé” y un id del paciente a procesar en caso de necesitarse datos anteriores.

```
def receive  ={

  case TramaP(t,id) =>
```

Dentro de dicha función se pueden definir los filtros de tal manera que el resultado sea un arreglo de números dobles, el cual se mandara por mensaje al actor que solicito dicho procesamiento.

La estructura de un filtro luce de la siguiente manera:




Si se quiere encadenar a los filtros existentes además de incluir la función del tipo anterior debe llamarse invocarse y guardarse el resultado como se muestra en la última línea del anterior fragmento de código.

El anterior filtro no depende de los datos anteriormente procesados del paciente, para incluir un filtro de esta índole debe tenerse en cuenta el HashMap de usuarios

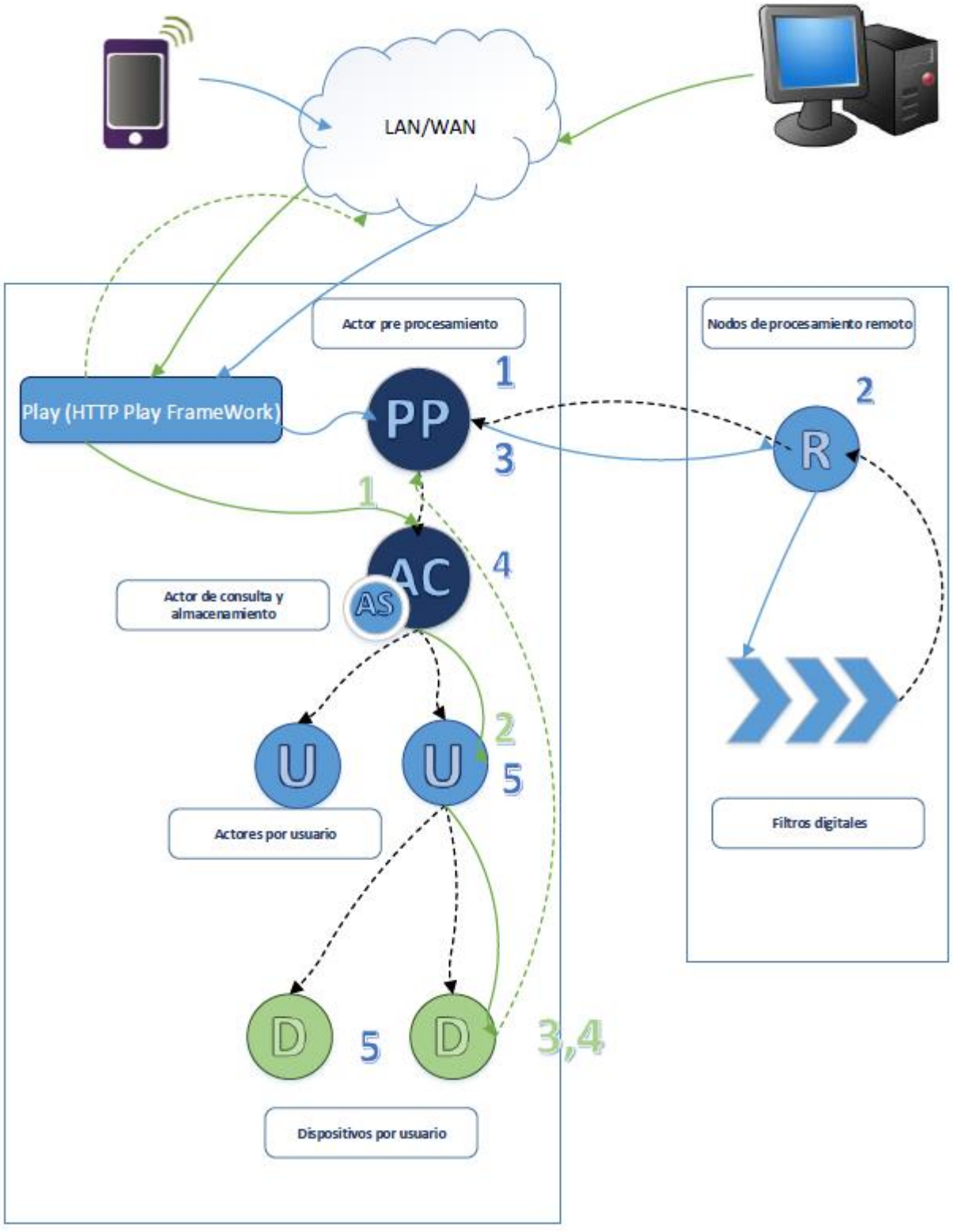
donde la llave es el ID del usuario actual y el valor, un arreglo de los últimos datos procesados. De la siguiente manera (asumiendo que los datos del usuario ya están en el HashMap):

```
if(datos.isEmpty)datos=t
var tramA:Array[Double]= datos.toArray
var tramAc:Array[Double]= t.toArray
var resul:List[Double]=List()
for(a <-1 to tramA.length){
  var valor = tramA(a)+ tramAc(a)*tramA(tramA.length-a) - tramAc(tramA.length-a) + tramAc(tramAc.length-1)
  resul=valor::resul
}
datos=resul
```

Vale la pena tener en cuenta, que si se quiere usar dichos cambios en el procesador remoto deben de hacerse los mismos cambios en la carpeta “Remote” del directorio “Código fuente”, específicamente en el directorio “src/main/scala/model” de dicha carpeta.

# Estructura de la aplicación

La arquitectura presentada en la figura a continuación representa el funcionamiento de la aplicación en dos escenarios:



- Escenario de transmisión y registro de la señal generada por el sensor ECG y transmitida por el móvil:

  Se recibe la señal en un actor de pre procesamiento (1), se envía al actor de procesamiento, que devolverá la trama después de la aplicación de varios filtros, tal como se muestra en el algoritmo (2). La trama procesada vuelve al actor de pre procesamiento para realizar el proceso de almacenado (3), posteriormente la trama es enviada al actor de consulta y guardado (4) y finalmente Si el paciente no tiene datos registrados en el sistema se crea un nuevo actor que represente dicho paciente y se almacenan los enviados. De estar registrado se actualizan todos los actores que representan a aquellos dispositivos consultando al paciente(5)

- Escenario de consulta de una señal procesada por parte de un cliente:

  El actor de consulta recibe la petición del dispositivo móvil o navegador de consultar una petición (1), en caso de existir un paciente identificado con el identificador proporcionado se procede a enviar una petición de consulta a dicho actor (2).Estando en el actor representante del paciente, este obtiene el identificador del dispositivo que solicita la consulta de datos para determinar si ya está registrado. En caso de no estarlo se crea un nuevo actor que represente dicho dispositivo y se devuelve la última trama de datos registrada del paciente (3).Finalmente El actor representante del dispositivo devuelve al actor de consulta principal la trama de datos más reciente no consultada de dicho dispositivo. Entendiendo que este mantiene una lista de tramas que va siendo consumida cada vez que el dispositivo en cuestión haga una consulta, manteniendo de esta manera la continuidad así halla perdidas de Internet o se corte la solicitud de datos por periodos cortos de tiempo (4).