Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Aplicación de envío de alertas basado en SCAIP

Autor: Germán Blas Aguilar Rodríguez
Tutor: Fernando Cárdenas

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Aplicación de envío de alertas basado en SCAIP

Autor:
Germán Blas Aguilar Rodríguez

Tutor:
Fernando Cárdenas
Profesor asociado a tiempo parcial

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Proyecto Fin de Carrera: Aplicación de envío de alertas basado en SCAIP

Autor: Germán Blas Aguilar Rodríguez

Tutor: Fernando Cárdenas

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2023

El Secretario del Tribunal

A mis amigos de toda la vida, Pablo y Guillermo, que han estado conmigo desde que tengo uso de razón.

A mis compañeros del voleibol, Carlos, Ninfa, Pablo, Sheila y Javubu, por ese espacio de risas y descanso.

A mis colegas de la carrera, Muni, Enrique, Marco y Dani, por quedarnos sin clases que saltar.

A los HitAnos. Teketeki, Jaime, Piñe, Fernando, Antonio (sin gluten), Felipe, Juan, y Avibi, no liberen el chat de Discord.

A los que no nombro, pues vuestros nombres son demasiados, pero ninguno menos importante.

A Cristina, por acompañarme en este viaje llamado vida.

A mi familia, a los que pueden ver lo que he conseguido y a los que no, por el cariño que me han dado.

A mí, por cada paso que he dado, por cada paso que seguiré dando, por cada paso que das.

Y a mi madre. Soy sangre de tu sangre. Gracias por hacerme quien soy. Te quiero.

Resumen

El objetivo de este trabajo es profundizar en los conocimientos sobre el diseño de aplicaciones móviles mediante la creación de una aplicación móvil que simule un dispositivo asistencia social, centrándose en el uso y comprensión de un protocolo de telecomunicaciones de asistencia social llamado SCAIP.

Este proyecto se ha dividido en seis secciones:

La primera sección es una breve introducción que establece el propósito y los objetivos que se persiguen en esta memoria.

A continuación, se presenta una sección de investigación teórica, donde se abordan los conceptos necesarios para llevar a cabo el trabajo de manera adecuada.

Posteriormente, se presenta una sección que describe la arquitectura del escenario y se realiza una investigación sobre las herramientas necesarias para llevar a cabo este trabajo.

La cuarta sección se centra en la arquitectura de la solución desarrollada y explica el uso de cada clase implementada.

La quinta sección aborda las pruebas que se llevaron a cabo para garantizar la correcta ejecución de nuestra aplicación.

Finalmente, la última sección concluye la memoria con una evaluación del desarrollo de este trabajo, resaltando los problemas encontrados y las lecciones aprendidas a lo largo del proceso.

Abstract

The objective of this work is to deepen the knowledge of mobile application design by creating a mobile application that simulates a social care device, focusing on the use and understanding of a social care telecommunications protocol called SCAIP.

This project has been divided into six sections:

The first section is a brief introduction that establishes the purpose and objectives pursued in this report.

This is followed by a theoretical research section, where the concepts necessary to carry out the work properly are addressed.

This is followed by a section describing the architecture of the scenario and an investigation of the tools needed to carry out this work.

The fourth section focuses on the architecture of the developed solution and explains the use of each implemented class.

The fifth section discusses the tests that were carried out to ensure the correct execution of our application.

Finally, the last section concludes the report with an evaluation of the development of this work, highlighting the problems encountered and the lessons learned throughout the process.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

El objetivo de este trabajo es desarrollar una aplicación móvil que simule un dispositivo de teleasistencia que genere alarmas concordes al estándar SCAIP. El estándar SCAIP (Social Care Alarm Internet Protocol) establece normas para la gestión de comunicaciones multimedia entre varios dispositivos de teleasistencia.

En la actualidad, los dispositivos multimedia están presentes en todos los aspectos de nuestra vida, de diversas formas y configuraciones. Por eso mismo, una buena estrategia será desarrollar una aplicación de software libre capaz de funcionar en dispositivos Android, ya que esto garantiza que funcione con la mayoría de los dispositivos disponibles. Específicamente, nuestra aplicación estará orientada a funcionar en relojes inteligentes, dado que son dispositivos fáciles de adquirir, tienen sensores apropiados para las personas mayores y se mantienen constantemente en sus manos sin causar molestias.

En este trabajo se realizará un análisis teórico que abarcará el estudio del estándar SCAIP, el protocolo de telecomunicaciones SIP (Session Initiation Protocol) en el que se basa, la librería seleccionada para su implementación y el software necesario para implementar un proxy SIP. Posteriormente, se llevará a cabo una presentación del escenario usado para realizar pruebas para verificar el correcto funcionamiento de la aplicación desarrollada junto a un análisis práctico del del código de la aplicación. Por último, se hará un análisis de los resultados de las pruebas ejecutadas y una evaluación final del trabajo para ver los objetivos logrados.

El resultado final de este trabajo busca adquirir conocimientos y experiencia en el proceso de desarrollo de aplicaciones móviles, así como enfrentarse a los desafíos y aprendizajes que ello conlleva.

2 TEORÍA

En la siguiente sección, realizaremos un análisis teórico para adquirir los conceptos fundamentales necesarios en el desarrollo de nuestra aplicación móvil. Nos enfocaremos en el estudio del estándar SCAIP y el protocolo SIP con el propósito de obtener los conocimientos necesarios para aplicar estas tecnologías en nuestro trabajo.

2.1. Protocolo de telecomunicación SIP

2.1.1 Introducción

En el ámbito de las telecomunicaciones existe una amplia variedad de aplicaciones que requieren la creación y gestión de sesiones, así como el intercambio de datos entre participantes.

El protocolo SIP se encarga de establecer, modificar y finalizar sesiones multimedia o llamadas que involucran a uno o más participantes. Estas sesiones pueden incluir llamadas telefónicas, distribución de multimedia, conferencias multimedia y otras aplicaciones similares. SIP puede invitar tanto a personas como servidores multimedia a unirse a las sesiones. Además, SIP se puede usar para iniciar sesiones o agregar participantes a sesiones que se han anunciado a través de otros medios, como el protocolo SAP(Session Announcement Protocol) [1].

Es importante destacar que SIP no ofrece servicios específicos; en cambio, proporciona primitivas que pueden utilizarse para implementar una variedad de servicios. Para la localización de los participantes de una sesión y otras funciones, SIP permite la creación de una infraestructura de servidores proxy en la red, donde los agentes pueden enviar registros, invitar a sesiones y realizar otras solicitudes.

2.1.2 Definiciones

Cliente: Un cliente es cualquier elemento de red que envía peticiones SIP y recibe respuestas SIP.

Conferencia: Una conferencia es una sesión multimedia que puede tener desde cero hasta varios miembros, incluyendo conferencias multicast o llamadas entre dos personas.

Diálogo: Un diálogo es una relación punto a punto entre dos clientes que persiste durante cierto tiempo. Se establece mediante mensajes SIP, como una respuesta 2xx a una solicitud INVITE. Un diálogo se identifica por un Call-ID, una etiqueta local y una etiqueta remota.

Invitación: Una invitación es una solicitud enviada por un usuario para unirse a una sesión. Una invitación exitosa consta de una solicitud INVITE terminada con un ACK.

Llamada: Una llamada es una conferencia que incluye a todos los participantes invitados por la misma fuente. Una llamada SIP se identifica mediante un Call-ID globalmente único. Si un usuario es invitado a la misma sesión multimedia a través de diferentes medios, los Call-ID de cada invitación serán diferentes.

Servidor: Un servidor es una aplicación que acepta solicitudes SIP (Request) para procesarlas y proporcionar el servicio correspondiente, luego envía respuestas (Response) a esas solicitudes. Un proxy es un ejemplo de servidor.

Sesión: Una sesión multimedia es un grupo de emisores y receptores de medios con su flujo de datos correspondiente.

Transacción: Una transacción ocurre entre un cliente y un servidor y comprende todos los mensajes desde la primera solicitud del cliente hasta la respuesta final del servidor al cliente. Una transacción se identifica mediante su número de secuencia, CSeq. La respuesta ACK tiene el mismo CSeq que la solicitud INVITE

correspondiente.

2.1.3 Funcionalidad del protocolo

SIP soporta cinco facetas en el establecimiento y terminación de comunicaciones multimedia.

Localización del usuario: Esta faceta se refiere a la capacidad de determinar la ubicación física o lógica de un usuario en el momento de establecer una sesión de comunicación. Los datos de ubicación, como la dirección IP del usuario o la dirección física del dispositivo, se pueden obtener a través de los mensajes SIP, que se guardan en una base de datos, o a través de consultas a servidores DNS para resolver dominios.

Capacidades del usuario: Esta faceta determina el contenido multimedia y los parámetros a utilizar en la comunicación. Estas capacidades se definen y se comunican entre dispositivos mediante el uso de encabezados específicos como "supported" o "require". Alguna de las capacidades definibles son el soporte para audio o video, capacidad de conferencia, transferencia de llamadas, códecs admitidos y funciones de seguridad.

Disponibilidad del usuario: Se refiere a la disposición de la parte llamada para participar en las comunicaciones. Esto facilita una correcta gestión de las comunicaciones y la toma de decisiones sobre cómo enrutar llamadas o mensajes. La disponibilidad del usuario se comunica mediante indicadores como "disponible", "ocupado", "ausente" o "desconectado" utilizando el encabezado "Contact" en los mensajes SIP.

Configuración de la llamada: Esta faceta abarca los parámetros y opciones que se utilizan para establecer y supervisar una sesión de comunicación en tiempo real. Esto incluye identificar a los participantes, elegir los medios de comunicación, establecer la calidad del servicio y brindar servicios adicionales como transferencia y conferencias.

Manejo de la llamada: Esta faceta abarca la transferencia y terminación de sesiones, así como la modificación de estas. Incluye funciones como el control de la sesión, la gestión de los medios, el mantenimiento de la sesión y la gestión de interrupciones.

2.1.4 Arquitectura del protocolo

El protocolo SIP se sustenta sobre la capa de transporte y admite diferentes protocolos, como TCP y UDP. Cuando se requiere una entrega confiable de mensajes se utiliza TCP para garantizar que lleguen sin errores y en el orden correcto. Por otro lado, UDP se utiliza para mensajes que no requieren entrega confiable, priorizando la eficiencia y la velocidad de transmisión.

Además, SIP admite TLS como protocolo adicional de transporte. La autenticación y el cifrado de los datos transmitidos en la comunicación SIP brindan una capa adicional de seguridad a través de TLS. Esto garantiza que la información sensible, como las credenciales de usuario y otros datos confidenciales, esté protegida contra posibles amenazas y ataques.

Cabe mencionar que SIP utiliza puertos predeterminados para el transporte de mensajes, como el puerto 5060 para UDP y TCP, y el puerto 5061 para TLS.

Si bien SIP puede utilizarse por sí mismo para establecer y mantener sesiones multimedia, es común emplear protocolos adicionales para mejorar la calidad y eficiencia de estas sesiones. Algunos de estos protocolos son SDP, RTP y RTCP. Estos protocolos adicionales complementan SIP al permitir la negociación, el transporte, el control y la gestión de recursos, lo que resulta en sesiones multimedia de mayor calidad y eficiencia.

2.1.5 Mensajes SIP

El mensaje SIP es la unidad fundamental de intercambio de información en el protocolo SIP. Está compuesto por una línea inicial, encabezados y, opcionalmente, un cuerpo. El mensaje SIP lleva consigo datos relevantes para establecer, modificar o finalizar una sesión de comunicación en redes IP.

2.1.5.1 Petición SIP

SIP utiliza diferentes métodos en las peticiones SIP para intercambiar información y llevar a cabo acciones específicas. Los seis métodos principales son:

INVITE: Este método se utiliza para iniciar una sesión. El cliente SIP envía una invitación al servidor SIP, especificando los parámetros de la sesión, como las direcciones IP y los códecs a utilizar.

ACK: Este método se utiliza para confirmar la recepción exitosa de una respuesta final, como 200 OK, a la solicitud INVITE.

BYE: Este método se utiliza cuando uno de los clientes SIP solicita la terminación de la sesión.

OPTIONS: El método se utiliza para obtener información sobre las capacidades y configuración de un servidor SIP o de un usuario registrado en el servidor.

REGISTER: Este método se utiliza para que un cliente SIP se registre en un servidor SIP. Permite al cliente anunciar su disponibilidad y proporcionar información de contacto, como su dirección IP y número de teléfono.

CANCEL: Este método se utiliza para cancelar una invitación INVITE que aún no ha sido respondida. Se envía desde el remitente al destino para cancelar la invitación pendiente.

Este séptimo método está indicado en una extensión de la norma SIP y es de interés ya que está fuertemente relacionado con el estándar SCAIP

MESSAGE: El método MESSAGE en SIP se utiliza para el envío de mensajes de texto entre usuarios o dispositivos. Este método proporciona una forma flexible de comunicación textual entre los participantes, brindando la posibilidad de intercambiar información de manera eficiente y sin interrumpir la sesión en curso[2].

2.1.5.2 Respuesta SIP

El equivalente a los métodos en las peticiones SIP son las líneas de estado que son los componentes que indican el estado o resultado de una solicitud realizada en el protocolo SIP. Estas líneas contienen un código de estado y una descripción textual que proporciona información sobre el resultado de la solicitud. Algunos de los códigos de estado comunes en las respuestas SIP incluyen:

1xx: Respuestas informativas que indican que la solicitud ha sido recibida y el servidor está procesando la solicitud. Algunos ejemplos son **100 Trying**, que indica que el proxy ha recibido la solicitud y está procesándola, o **180 Ringing**, que indica que el destinatario está siendo informado de una llamada entrante

2xx: Respuestas exitosas que indican que la solicitud se ha completado correctamente. Algunos ejemplos son **200 OK** que indicar que la solicitud ha sido procesada con éxito, o **202 Accepted**, que indica que la solicitud ha sido aceptada pero aún no ha sido procesada por completo.

3xx: Respuestas de redirección que indican que se requiere tomar acciones adicionales para completar la solicitud. Un ejemplo sería **301 Moved Permanently**, que indica que el recurso solicitado ha sido trasladado a una nueva ubicación

4xx: Respuestas de error del cliente que indican que ha ocurrido un error en la solicitud realizada por el cliente. Algunos ejemplos son **404 Not Found**, que indica que el recurso solicitado no ha sido encontrado, o **403 Forbidden**, que indica que el cliente no tiene permiso para acceder al recurso solicitado.

5xx: Respuestas de error del servidor que indican que ha ocurrido un error en el servidor al procesar la solicitud. Un ejemplo sería **503 Service Unavailable**, que indica que el servidor no está disponible para procesar la solicitud en ese momento.

2.1.5.3 Encabezados

Los encabezados de un mensaje SIP contienen información adicional que son necesarios para el procesamiento y control de la comunicación. Algunos de los encabezados más comunes en un mensaje SIP incluyen:

Via: Este encabezado indica la ruta que ha seguido el mensaje SIP a través de los proxies y servidores SIP desde el remitente hasta el destinatario. Cada proxy o servidor SIP agrega su dirección IP al encabezado Via, lo que permite el seguimiento de la ruta tomada.

From: Este encabezado identifica al remitente de la petición o respuesta SIP. Contiene la dirección URI del

remitente, que puede ser un número de teléfono, una dirección de correo electrónico o una dirección IP.

To: Este encabezado identifica al destinatario de la solicitud o respuesta SIP. Al igual que el encabezado From, contiene la dirección URI del destinatario.

Call-ID: Este encabezado es un identificador único para una llamada o sesión SIP. Permite a los dispositivos involucrados en la comunicación asociar múltiples mensajes relacionados con la misma llamada.

CSeq: El encabezado CSeq (Sequence Number) indica el número de secuencia del mensaje dentro de una llamada o sesión. Ayuda a mantener el orden de los mensajes y proporciona una forma de correlacionar las respuestas con las solicitudes correspondientes.

Content-Type: Este encabezado se utiliza cuando el mensaje SIP contiene un cuerpo, como una descripción de sesión en formato SDP. El encabezado Content-Type especifica el tipo de contenido y su formato.

Authorization: Este encabezado se utiliza para autenticar y autorizar a un usuario que realiza una solicitud SIP. Contiene credenciales de autenticación que se utilizan para verificar la identidad del remitente y permitir el acceso a los recursos protegidos.

Max-Forwards: El encabezado Max-Forwards especifica el número máximo de saltos que puede tomar un mensaje SIP antes de ser descartado.

Content-Length: Cuando un mensaje SIP tiene un cuerpo, el encabezado Content-Length indica la longitud del contenido en bytes.

2.1.6 Ejemplo de flujo

En la RFC 3261 del protocolo SIP, se presenta un ejemplo de flujo de mensajes que ilustra el intercambio de mensajes entre distintos usuarios, mostrando el proceso típico de establecimiento y finalización de una llamada utilizando SIP.

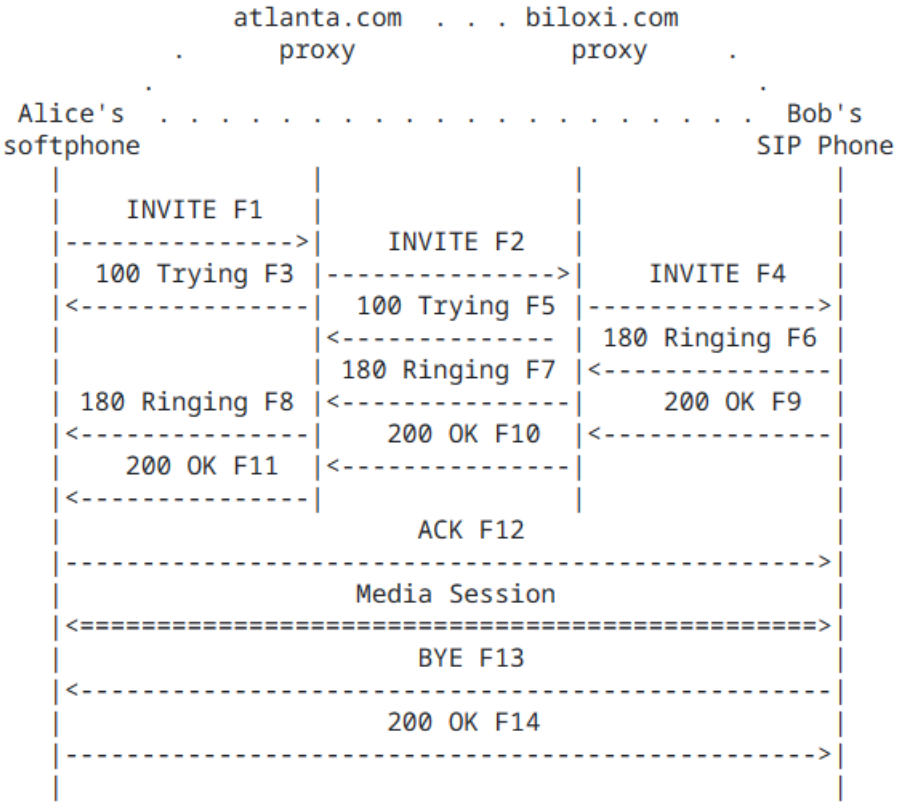

Figura 1. Diagrama de ejemplo de flujo de mensajes [1]

El flujo de mensajes comienza cuando el cliente envía una solicitud INVITE al servidor SIP para iniciar una llamada con otro cliente. Esta solicitud contiene la información del destino de la llamada. A medida que los

mensajes viajan entre los distintos servidores, lo más normal es que tengan que pasar por proxy intermedios que ayudan en el enrutamiento y procesamiento de las solicitudes y respuestas SIP.

El llamado responde con un mensaje de respuesta 100 Trying, que indica que la solicitud ha sido recibida y se están tomando las acciones necesarias. A continuación, envía una respuesta 180 Ringing al llamante, indicando que la llamada está sonando en el dispositivo del destinatario. Finalmente, el llamado envía una respuesta final 200 OK al llamante, confirmando que la llamada ha sido aceptada.

Para finalizar la llamada, cualquiera de los participantes puede enviar una solicitud BYE al otro. Una vez recibida, se responde con una respuesta 200 OK, confirmando el cierre de la sesión y la finalización de la llamada.

2.1.7 Seguridad en el protocolo SIP

Para proteger las comunicaciones y garantizar la confidencialidad, integridad y autenticidad de los mensajes SIP, el protocolo SIP ofrece varios mecanismos de seguridad. Los mecanismos de seguridad que vamos a implementar en este trabajo son:

TLS: SIP puede utilizar TLS para establecer una capa de seguridad adicional en el transporte de los mensajes SIP. El cifrado de extremo a extremo y la autenticación de los participantes en la comunicación proporcionados por TLS protegen los mensajes contra la interceptación y manipulación por parte de terceros no autorizados. Para ello, serán necesarios los certificados digitales. Estos certificados contienen una clave pública y una firma digital que garantiza la autenticidad y confiabilidad de la entidad que lo posee.

Autenticación Digest: La autenticación digest es un mecanismo de seguridad que verifica la identidad de los usuarios mediante un algoritmo criptográfico (MD5). El servidor SIP desafía al cliente con un nonce, un número único que se usa una vez durante el proceso de autenticación. El cliente calcula un valor hash usando el algoritmo criptográfico utilizando este nonce, junto con otros parámetros y la contraseña del usuario. La autenticación se considera exitosa si el valor hash coincide con el valor que calculó el servidor. El nonce evita el acceso no autorizado y garantiza que cada solicitud de autenticación sea única.

2.2 Estándar SCAIP

SCAIP es un estándar que establece un protocolo de comunicación para los servicios de teleasistencia. La estandarización de este protocolo abierto es necesaria para garantizar la interoperabilidad en el mercado de los servicios de teleasistencia.

SCAIP utiliza funciones basadas en SIP para la iniciación, direccionamiento y transporte de datos, permitiendo el establecimiento de flujos de medios y la transferencia de información entre el usuario y el receptor. El intercambio de datos del protocolo se define mediante un esquema XML que incluye tipos de alarmas, códigos e información adicional.

2.2.1 Términos y abreviaciones

Heartbeat (Latido): Evento periódico generado por hardware o software para indicar el funcionamiento normal o para sincronizar partes de un sistema.

Sistema de teleasistencia: Sistema que proporciona facilidades las 24 horas para la activación de alarmas, identificación, transmisión de señales, recepción de alarmas, comunicación bidireccional de voz, tranquilidad y asistencia, para ser utilizado por personas consideradas en situación de riesgo.

LUC (Local Unit and Controller): Emisor de alarmas

ARC (Alarm Receiving Center): Receptor de alarmas

2.2.2 Casos de uso

El estándar SCAIP describe dos casos de uso genéricos: uno sin el establecimiento de una sesión de voz o multimedia, y otro caso de uso que describe un intercambio de mensajes de evento combinado con una sesión de voz o multimedia adicional utilizando SIP.

2.2.2.1 Caso de uso 1: Evento sin sesión de voz o multimedia.

La Figura 1 muestra un modelo de comunicación con un emisor de alarmas que se comunica a través de la red utilizando SCAIP como protocolo y un receptor de alarmas como punto final. Los mensajes pueden ser iniciados tanto por un usuario final como por un dispositivo, y pueden tener la función de ser una alarma o un mensaje de estado.

Figura 2. Arquitectura de los casos de uso [3]

La figura 2 muestra un intercambio de mensajes entre el emisor de alarmas y el receptor de alarmas

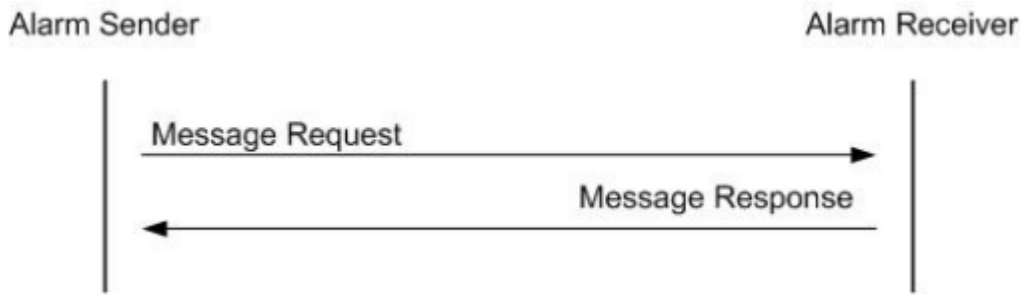

Figura 3. Diagrama de intercambio de mensajes de evento sin sesión de voz o multimedia [3]

En este caso de uso el emisor de alarmas envía un mensaje de petición al receptor de alarmas. Cada petición debe ser respondido por un mensaje de respuesta.

2.2.2.1.1 Heartbeat

Uno de los casos específicos de este caso de uso es el heartbeat (latido). En este caso, el emisor de alarmas debe realizar consultas periódicas al receptor de alarmas enviando un mensaje de petición con el tipo de mensaje establecido como latido. A su vez, el receptor de alarmas debe responder con un mensaje de respuesta en el que se establezca el número de estado en 0.

El período del latido deberá ser de al menos 1 minuto y como máximo 1440 minutos.

2.2.2.2 Caso de uso 2: Evento con sesión de voz o multimedia.

En este caso de uso, el emisor de alarmas enviará un mensaje de petición y establecerá una sesión de voz o multimedia. El intercambio de mensajes se manejará de la misma manera que en el caso de uso 1, con la excepción de que las sesiones de voz o multimedia solo deben ser iniciadas por el emisor de alarmas después de recibir un mensaje de respuesta con un número de estado establecido en 0 y una respuesta de medios establecida en 1 o más.

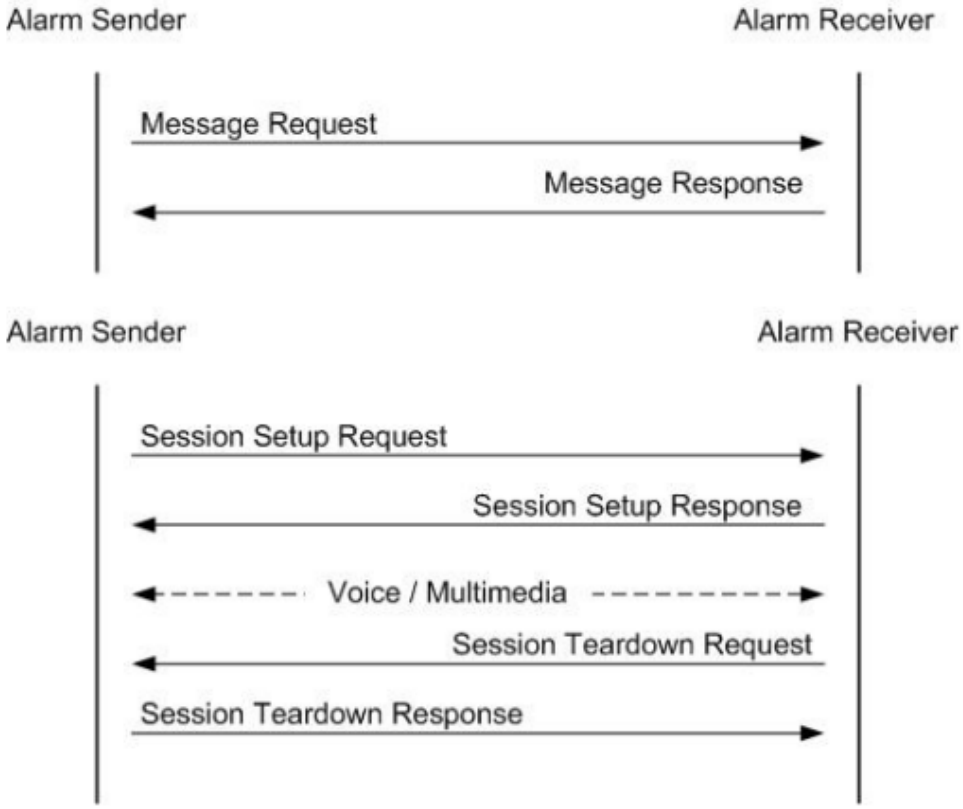

Figura 4. Diagrama de intercambio de mensajes de evento con sesión de voz o multimedia [3]

Una vez que se reciba el mensaje de respuesta, el emisor de alarmas deberá iniciar la sesión de voz en el margen de 60 segundos.

La comunicación de voz o multimedia deberá ser iniciada por el emisor de alarmas. Una vez que la comunicación esté establecida, podrá ser modificada o finalizada tanto por el emisor de alarmas como por el receptor de alarmas.

2.2.3 Formato de los mensajes

SCAIP usa un formato XML para los mensajes para representar las alarmas y los códigos de información. Estos mensajes deben conformarse con un esquema XML que deberá usarse para verificar los mensajes. Todo el contenido XML tiene que ser transportado por una petición SIP MESSAGE.

2.2.3.1 Message Request

El mensaje de petición debe ser codificado como una lista de etiquetas XML contenidas en la etiqueta <mrq> (message-request) de acuerdo con la plantilla siguiente. La platilla define la codificación XML de todos los posibles elementos.

```
<mrq>
    <ref>reference</ref>
    <ver>version</ver>
    <sco>system-config</sco>
    <cha>call-handling</cha>
    <mty>message-type</mty>
    <hbo>heartbeat-options</hbo>
    <cid>controller-id</cid>
    <dty>device-type</dty>
    <did>device-id</did>
    <dco>device-component</dco>
    <dte>device-text</dte>
    <crd>caller-id</crd>
    <stc>status-code</stc>
    <stt>status-text</stt>
    <pri>priority</pri>
    <lco>location-code</lco>
    <lva>location-value</lva>
    <lge>
         <geo>wgs-pos</geo>
         <tim>time-stamp</tim>
         <gga>gga-pos</gga>
    </lge>
    <lte>location-text</lte>
    <ico>info-code</ico>
    <lte>info-text</lte>
    <ame>additional-message</ame>
</mrq>
```

La siguiente tabla describe todos los posibles elementos y las reglas de construcción. Los campos con un * son mandatorios.

Campo	Etiqueta XML	Descripción del contenido	Longitud en caracteres
message-request	<mrq>	Elemento compuesto para el mensaje de petición	
reference	<ref>	La referencia se utiliza para comparar la respuesta del mensaje con la solicitud de mensaje. El número de referencia debe ser diferente en cada sesión de mensajes. Los caracteres válidos son: "A" a "Z", "a" a "z" y "0" a "9".	1-16 *

version	<ver>	La versión del protocolo se escribe como dos dígitos, un punto y dos dígitos. Ejemplo: "01.23" Cuando se omite, se asume que la versión es "01.00". Los caracteres válidos son "0" a "9".	0/5
system-config	<sco>	Configuración e información del sistema: Omitido o "0" = Unidad local y controlador "1" = Equipos agrupados con supervisor fuera de servicio "2" = Equipos agrupados con supervisor de servicio "3" = Equipos agrupados con supervisor de servicio actuando como receptor de alarmas.	0/1
call-handling	<cha>	Omitido o "0" = Llamada saliente normal desde el remitente de la alarma "1" = Se solicita una llamada de devolución.	0/1
message-type	<mty>	Omitido o "ME" = Mensaje "RE" = Reinicio "IN" = Actualización de información dentro de la sesión de mensajes "PI" = Latido (Heartbeat)	0/2
heartbeat-options	<hbo>	Omitido o "0" = el remitente de la alarma no puede manejar ajustes de tiempo de latido (Heartbeat) del receptor de la alarma. "001" = el remitente de la alarma puede manejar ajustes de tiempo de latido del receptor de la alarma. Otros = no especificado para uso futuro.	0/1-3
controller-id	<cid>	ID o número de cuenta para el controlador que está realizando la llamada. Los caracteres válidos para este campo son "0" a "9".	1-32*
device-type	<dty>	El tipo de dispositivo contiene la especificación del dispositivo. Los tipos de dispositivo válidos se definen en el anexo A de la especificación SCAIP.	1-4*
device-id	<did>	El ID del dispositivo se utiliza para indicar la identidad de un dispositivo físico que causa el evento. Los caracteres válidos para este campo son caracteres con codificación US-ASCII del 32 al 126.	0/1-16
device-component	<dco>	El componente del dispositivo se utiliza para acotar más la fuente de un evento dentro de un dispositivo. Los códigos válidos se definen en el anexo B de la especificación SCAIP.	0/1-3
device-text	<dte>	Descripción de texto del dispositivo. El texto tiene como objetivo proporcionar información adicional, sin contradecir el tipo o componente del dispositivo. El conjunto de caracteres válido para este campo es Unicode, y se debe utilizar la codificación UTF-8.	0/1-32
caller-id	<crd>	Media y número de teléfono o SIP URI del remitente de la alarma. Ejemplo 1: "gsm:4671445566" Se debe utilizar un elemento XML para cada medio disponible. Los tipos de medios válidos son: "no-voice:" "gsm:" "sip:" "sip-pp:" Si el número de teléfono o SIP URI es desconocido, pero se puede utilizar para llamadas salientes, se debe omitir el número. Omitido = "sip:" Ejemplo 2: "gsm:"	0/1-256
status-code	<stc>	El código de estado se utiliza para indicar los estados del dispositivo. Los códigos de estado válidos se definen en el anexo C de la documentación de SCAIP. Cuando se omite, se asume el estado de alarma.	0/1-4

status-text	<stt>	Descripción de texto del estado. El texto solo debe proporcionar información adicional y no debe contradecir el código de estado. El conjunto de caracteres válido para este campo es Unicode y se debe utilizar la codificación UTF-8.	0/1-32
priority	<pri>	Un campo de prioridad de un solo dígito para indicar que se ha asignado una prioridad sobre otros mensajes a este evento. "0" = prioridad más baja "1"-"8" = prioridad creciente "9" = prioridad más alta Omitido = "0"	0/1
location-code	<lco>	Información de ubicación para permitir indicar ubicaciones detalladas de eventos. El código de ubicación también puede hacer referencia a un valor de ubicación cuando sea necesario. Los códigos de ubicación válidos se definen en el anexo D de la documentación de SCAIP.	0/1-3
location-value	<lva>	Especificación adicional del código de ubicación. Puede ser tanto una numeración simple de una ubicación fija, un área o un número de sección. Los caracteres válidos para este campo son caracteres con codificación US-ASCII del 32 al 126.	0/1-2
location-geo	<lge>	Cuando se agregue una coordenada geográfica o una marca de tiempo se deben utilizar los siguientes subelementos adicionales.	
wgs-pos	<geo>	Microformato 2D de sistema de referencia de coordenadas WGS-84 expresado como latitud y longitud en grados, extraído de "geo" especificado en RFC 5870. Ejemplo: "59.336111,18.072778"	0/1-23
time-stamp	<tim>	Marca de tiempo adicional según ISO 8601 con la zona horaria expresada como la diferencia con UTC en horas. Marca de tiempo en Hora de Europa Central (CET): YYYY-MMDDThh:mm:ss+01	0/22
gga-pos	<gga>	Sentencia NMEA de GPS, GGA - datos esenciales de fijación que proporcionan datos de ubicación y precisión en 3D. $GPGGA, hhmmss.ss, llll.ll, a, yyyyy.yy, a, x, xx, x.x, x.x, M, x.x, M, x.x, xxxx*hh	0/81
location-text	<lte>	Descripción de texto de la ubicación desde donde se origina la solicitud de mensaje. Ejemplo: "Chapel" El texto tiene como objetivo proporcionar información adicional, sin contradecir el código o valor de ubicación. Se debe utilizar el conjunto de caracteres válido Unicode y la codificación UTF-8.	0/1-32
info-code	<ico>	Información adicional codificada sobre el evento. Los códigos de información válidos se definen en el anexo E de la documentación SCAIP.	0/1-3
info-text	<ite>	Información de texto adicional sobre el evento. El texto tiene como objetivo proporcionar información adicional, sin contradecir el código de información. Se debe utilizar el conjunto de caracteres válido Unicode y la codificación UTF-8.	0/1-128
additional-message	<ame>	Se utilizará un mensaje adicional si la longitud total de la solicitud de mensaje excede los 1300 bytes. Si la solicitud de mensaje excede el límite, se enviará como varias solicitudes de mensaje marcadas con el número de mensajes adicionales después del mensaje actual. Ejemplo: si una solicitud de mensaje tiene 1700 bytes, se dividirá en dos mensajes, indicando en el primer mensaje que hay 1 mensaje adicional después del mensaje actual, y en el segundo (último) mensaje se indicará que no hay mensajes adicionales.	0/1-4

		Omitido o "0" = Último mensaje "n" = Número de mensajes adicionales después del mensaje actual Los caracteres válidos para este campo son "0" a "9".	

Tabla 1. Elementos XML del mensaje de solicitud SCAIP [3]

A continuación se presenta un ejemplo de alarma de solicitud SCAIP de tipo Heartbeat

```
<mrq>
    <ref>Heart001</ref>
    <mty>PI</mty>
    <hbo>001</hbo>
    <cid>836710811199107</cid>
    <dty>0002</dty>
</mrq>
```

2.2.3.2 Message Response

El mensaje de respuesta debe ser codificado como una lista de etiquetas XML contenidas en la etiqueta message-response de acuerdo con la plantilla siguiente.

```
<mrs>
    <ref>reference</ref>
    <snu>status-number</snu>
    <ste>status-text</ste>
    <cre>callhandling-reply</cre>
    <tnu>transferred-number</tnu>
    <hbi>heartbeat-interval</hbi>
</mrs>
```

Nombre del dato element	Etiqueta XML	Descripcion del contenido	Longitud en caracteres
message-response	<mrs>	Elemento compuesto para la Respuesta de Mensaje.	
reference	<ref>	El número de referencia extraído de <mrq>.	1-16*
status-number	<snu>	Números de estado relacionados con el procesamiento de alarmas: "0" = OK. La alarma se ha gestionado correctamente. No se necesitan más acciones. "4" = En espera. La alarma se está procesando. Vuelva a enviar la alarma para obtener información de estado actualizada. "5" = No tratada o no distribuida. No se pudo manejar la alarma. Intente otra acción de alarma. "6" = Ocupado. No se pudo manejar la alarma en este momento. Inténtelo nuevamente más tarde o continúe con la siguiente alarma. Números de estado relacionados con mensajes: "1" = mensaje demasiado largo. "2" = formato no válido. "3" = contenido de datos incorrecto. "7" = falta etiqueta obligatoria. "10" = falta referencia. "99" = error indefinido.	1-3*
status-text	<ste>	Información de texto sobre el número de estado. El texto tiene como objetivo proporcionar información adicional, sin contradecir el número de estado. Los caracteres válidos para este campo son caracteres con codificación US-ASCII del 32 al 126.	0-128

common-version	<cve>	La versión del protocolo a utilizar debe ser igual o inferior a la versión en la Solicitud de Mensaje correspondiente. Omitido = Versión inicial, "01.00" Los caracteres válidos son "0" a "9".	0/5
media-reply	<mre>	Omitido o "0" = sin llamada de voz "1" = llamada de voz dúplex "2" = llamada de voz simple, micrófono (altavoz LUC silenciado) "3" = llamada de voz simple, altavoz (micrófono LUC silenciado)	0/1-2
callhandling-reply	<cre>	Respuesta a la solicitud de manejo de llamada, <cha>. "01"-"60" (nn) = se acepta la devolución de llamada, se abre un canal de respuesta automática durante nn minutos. Requiere que el receptor de la alarma haya registrado el URI/número de teléfono válido del remitente de la alarma o un número o identidad contenido en el identificador de llamadas, <crd>. Omitido o "61" = establecer una llamada de voz a un receptor predefinido. "62" = establecer una llamada de voz al URI/número de teléfono transferido definido en el número transferido, <tnu>.	0/2
transferred-number	<tnu>	Medio y número de teléfono o URI SIP transferido para la respuesta de manejo de llamada, <cre>. EJEMPLO: "gsm:+46701445566" Se debe utilizar un elemento XML para cada medio disponible. Los tipos de medios válidos son: "gsm:" "sip:" "sip-pp:"	0/1-256
heartbeat-interval	<hbi>	Intervalo de latido El receptor de la alarma puede ajustar el intervalo de latido del remitente de la alarma si el remitente de la alarma ha indicado que puede manejar dicho ajuste configurando las opciones de latido <hbo> en "001". "1"-"1440" = Tiempo en minutos entre latidos consecutivos. Omitido = se utilizará el valor predeterminado especificado por el remitente de la alarma.	0/1-4

Tabla 2. Elementos XML del mensaje de respuesta SCAIP [3]

A continuación se muestra el mensaje de respuesta SCAIP básica exitosa al mensaje de ejemplo anterior

```
<mrs>
    <ref>Heart001</ref>
    <snu>0</snu>
</mrs>
```

2.2.4 Autentificación y encriptación

De acuerdo con la documentación del estándar SCAIP, tanto el emisor como el receptor de alarmas deben implementar un mecanismo de autenticación que sea compatible con la autenticación HTTP Digest. Además, es necesario que tanto el emisor como el receptor de alarmas cuenten con capacidades de cifrado. Para asegurar la privacidad de la sesión SIP, esta debe ser cifrada utilizando TLS v1.2 o una versión superior, y se deben emplear algoritmos criptográficos con un cifrado AES-128 mínimo [4].

3 INGENIERÍA

En esta sección, describiremos la arquitectura de nuestro escenario y examinaremos las herramientas clave que utilizaremos.

3.1 Arquitectura del trabajo

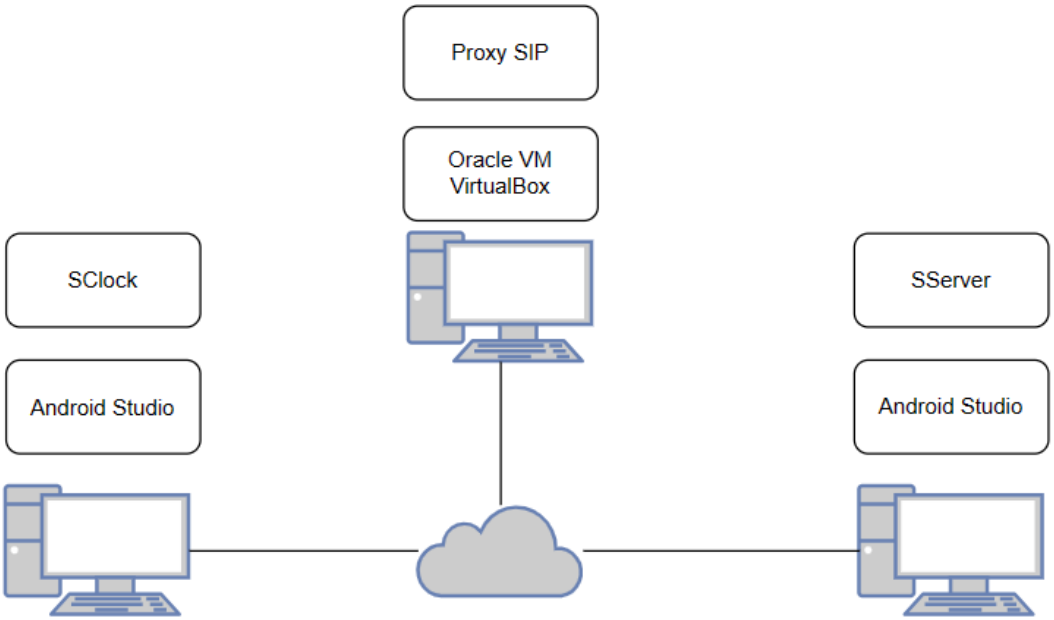

Figura 5. Arquitectura del trabajo

La solución propuesta consiste en dos aplicaciones principales:

Nuestra aplicación principal, SClock, es una aplicación Android para relojes inteligentes desarrollada en Android Studio, que va a actuar de cliente SCAIP. Esta aplicación se ejecutará en el emulador interno de Android Studio, de tipo Wear OS Large Round con versión API 30

Para lograr un escenario dinámico, se ha creado una aplicación secundaria, SClock, que actuará como servidor SCAIP y seguirá la misma estructura que la aplicación principal. Esta aplicación recibirá las peticiones SCAIP generadas por nuestra aplicación principal y generará una respuesta SCAIP para que pueda procesarla nuestro cliente.

Además, se configurará un proxy SIP en la máquina virtual proporcionada por la Universidad de Sevilla y se ejecutará en Oracle VM. Este proxy nos proporcionará funcionalidad de registro de usuarios SIP y enrutamiento. El siguiente enlace contiene las instrucciones para obtener la máquina virtual de la universidad de Sevilla: http://trajano.us.es/~fjfj/mv/maq_virtual.html.

El escenario se implementará en una LAN privada a través de un rúter privado que proporcionará direcciones IP dinámicas a los dispositivos conectados, lo que puede generar diferencias de IP en pruebas posteriores.

3.2 Librería JAIN SIP

3.2.1 Análisis de soluciones

En el mercado hay una amplia variedad de bibliotecas de software libre para implementar el protocolo SIP. Durante nuestro análisis, nos enfocamos en encontrar una solución que priorizara la robustez, facilidad de implementación y disponibilidad de documentación. De estas soluciones hemos decidido analizar las tres más usadas.

Características	JAIN-SIP-RI	PJSIP	Liblinphone (Linphone)
Lenguaje de programación	Java	C/C++, con wrappers para Java/Kotlin	C/C++, con wrappers para Java/Kotlin
Primera versión	Enero, 2001 (JAIN SIP)	Febrero, 2005	2001 (Linphone)
Ultima versión	1.3.0-91 23 de Marzo, 2018 (JAIN-SIP-RI)	2.13 24 de Noviembre, 2022	5.2.77 21 de Junio, 2023 Nuevas versiones constantemente
Modelo de programación	Orientada a eventos	Orientada a eventos	Orientada a eventos
Compatibilidad	Multiplataforma	Multiplataforma	Multiplataforma
Robustez	Alta robustez	Alta robustez	Alta robustez
Tipo de implementación	Librería de código abierto	Librería de código abierto	Librería de código abierto
Documentación y recursos	Amplia documentación	Amplia documentación	Amplia documentación
Soporte SIP	Cumple con los estándares SIP	Cumple con los estándares SIP	Cumple con los estándares SIP
Integración de códecs	Requiere bibliotecas externas	Incluye códecs de audio y video	Incluye códecs de audio y video
Comunidad y soporte	Comunidad y soporte activos	Comunidad y soporte activos	Comunidad y soporte activos
Seguridad	Soporte para cifrado de comunicaciones	Soporte para cifrado de comunicaciones	Soporte para cifrado de comunicaciones

Tabla 3. Análisis de soluciones de librerías SIP [5] [6] [7]

Como podemos observar, estas tres soluciones son excelentes debido a que todas ellas son robustas, ampliamente probadas, ofrecen funcionalidad completa de SIP, son de código abierto y están disponibles para una aplicación Android.

La elección de utilizar JAIN SIP en este trabajo se debe en parte a que es una API escrita en Java y usaré el mismo lenguaje para desarrollar la aplicación Android. Además, tiene la ventaja de contar con una amplia cantidad de ejemplos de uso, mientras que para las otras dos opciones me ha resultado más difícil encontrar

una cantidad similar de ejemplos. Para la implementación en este trabajo vamos a hacer uso de la librería JAIN SIP RI (Reference Implementation), que ofrece una implementación de esta API.

PJSIP y Liblinphone siguen siendo opciones muy sólidas, ya que además de estar en constante actualización, también cuentan con soporte integrado de audio y video. Sin embargo, para este trabajo en particular no se requieren funcionalidades tan avanzadas, ya que solo vamos a abordar el primer caso de uso, sin intercambio de datos de voz.

3.2.2 Introducción

JAIN SIP es una API para el desarrollo de aplicaciones de telecomunicaciones basadas en el protocolo SIP.

Esta API se crea para abordar la necesidad de implementar el protocolo SIP en el desarrollo de aplicaciones. Al estandarizar la interfaz, JAIN SIP garantiza que las aplicaciones desarrolladas con esta API puedan funcionar correctamente con diferentes implementaciones de pilas SIP, lo que facilita la integración y la comunicación entre sistemas heterogéneos.

JAIN SIP está destinado a los desarrolladores que requieren un acceso robusto al protocolo SIP. Puede usarse en una variedad de contextos, como proxy, agente de usuario o integrado en contenedores de servicios.

Además, se proporciona una librería que contiene una implementación de referencia, JAIN SIP RI (Reference Implementation), que utilizaremos en este proyecto[8].

3.2.3 Arquitectura de la librería JAIN SIP

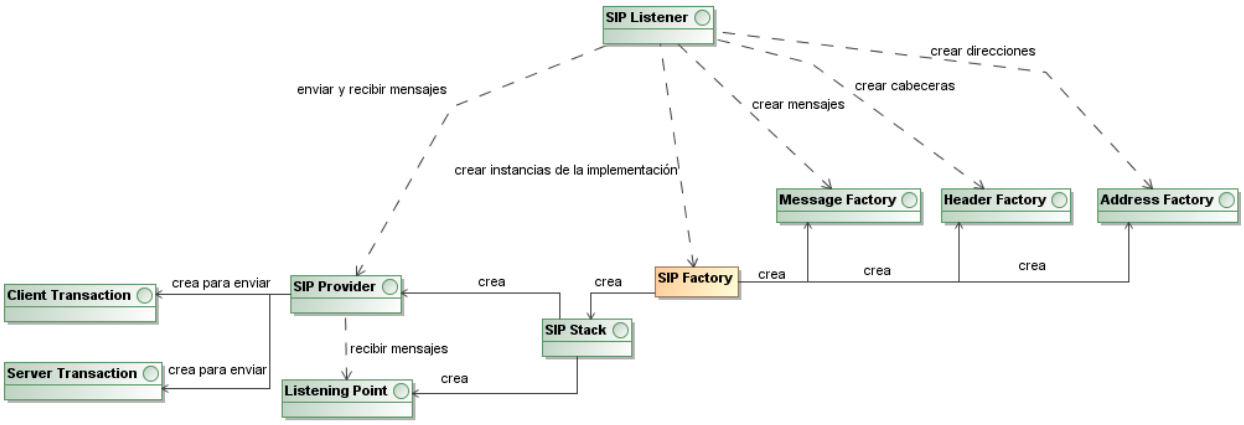

Figura 6. Arquitectura de la librería JAIN SIP

La arquitectura de la biblioteca JAIN SIP se compone de varios objetos clave que interactúan entre sí para brindar la funcionalidad de mensajería SIP. Estos objetos son:

3.2.3.1 SIP Factory

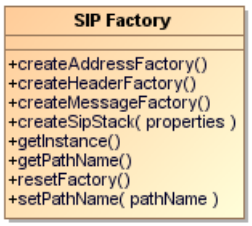

Figura 7. Diagrama de clase de SIP Factory

El objeto SIP Factory es el punto de acceso único para la fabricación de los objetos relacionados con la

implementación de la API JAIN SIP. La arquitectura de la implementación de JAIN SIP se guarda en un paquete raíz, llamado pathname. En este objeto establecemos el pathname de la implementación de referencia, que en nuestro caso es "android.gov.nist", para que indicar a la fábrica dónde están las implementaciones. El nombre de estas implementaciones tiene que agregar al final de sus nombres el sufijo "Impl" (por ejemplo, SipStack y SipStackImpl).

Esta factoría produce otras tres factorías que usamos para diferentes funciones: Address Factory (que define métodos para crear URIs SIP), Header Factory (que define métodos para crear los encabezados) y Message Factory (que define métodos para crear objetos de peticiones y respuestas).

3.2.3.2 SIP Listener

Figura 8. Diagrama de clase de SIP Listener

El SIP Listener es responsable de recibir y procesar eventos relacionados con la señalización SIP. Actúa como un receptor de mensajes SIP entrantes a través de eventos emitidos por el SIP Provider. Esta interfaz representa la vista de la aplicación hacia una pila SIP y, por lo tanto, define el canal de comunicación de la aplicación con la pila SIP.

Los eventos aceptados por un SIP Listener pueden ser:

RequestEvent: Este evento representa mensajes de solicitud, como INVITE, que son recibidos desde la red por la aplicación a través de la implementación de la pila subyacente.

ResponseEvent: Este evento representa mensajes de respuesta, como respuestas 2xx, que son recibidos desde la red por la aplicación a través de la implementación de la pila subyacente.

TimeoutEvent: Este evento representa la expiración de temporizadores en el SIP Provider. Estos eventos de tiempo de espera notifican a la aplicación que se requiere una retransmisión o que una transacción ha expirado.

IOExceptionEvent: Este evento representa un fallo en la capa de E/S subyacente del SIP Provider. Estos eventos de excepción de E/S notifican a la aplicación que ha ocurrido un fallo al acceder a un socket.

TransactionTerminatedEvent: Los eventos de TransactionTerminated representan la terminación de una transacción y notifican a la aplicación sobre esta.

DialogTerminatedEvent: Los eventos de DialogTerminated representan la terminación de un diálogo y notifican a la aplicación sobre esta.

3.2.3.3 SIP Stack

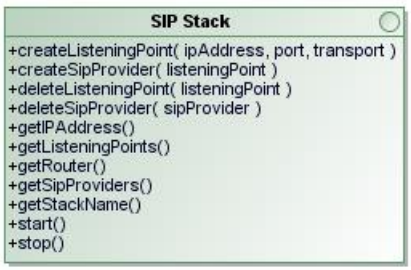

Figura 9. Diagrama de clase de SIP Stack

La interfaz SIP Stack representa la interfaz de gestión de una pila SIP que implementa JAIN SIP. Esta interfaz define los métodos que deben ser utilizados para controlar la arquitectura y configuración de la pila SIP. Estos métodos incluyen la creación y eliminación de los SIP Provider y los Listening Point.

3.2.3.4 SIP Provider

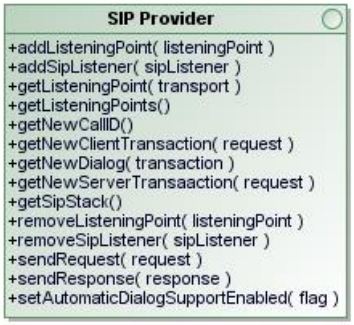

Figura 9. Diagrama de clase de SIP Provider

El SIP Provider representa el punto de entrada para la interacción con otros sistemas y redes SIP. Esta interfaz es la encargada de enviar y recibir mensajes SIP, establecer y terminar sesiones, y gestionar la señalización y el intercambio de información entre los participantes de una comunicación. Esta interfaz define los métodos que habilitan al SIP Listener, entre ellos:

Registrar un SIP Listener en el SIP Provider. Una vez que el SIP Listener está registrado en el SIP Provider, este recibirá notificaciones a través de eventos de los mensajes de petición, mensajes de respuesta, expiración de temporizadores, errores de red y terminación de transacciones y diálogos.

Cancelar el registro de un SIP Listener del SIP Provider. Una vez que se cancela el registro de un SIP Listener, ya no recibirá ningún evento de ese SIP Provider.

Crear objetos Client y Server Transaction para enviar mensajes con estados.

Enviar mensajes de petición y respuesta sin estado.

3.2.3.5 Listening Point

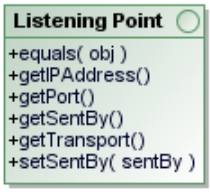

Figura 10. Diagrama de clase de Listening Point

Esta interfaz representa un único punto de escucha en una red IP, compuesto por el protocolo de transporte usado, el puerto y la dirección IP. Un Listening Point es una representación en Java del socket que utiliza una entidad de mensajería SIP Provider para enviar y recibir mensajes. Los Listening Point del SIP Provider especifican la dirección local desde donde se enviarán los mensajes de solicitud.

3.2.3.6 Client y Server Transaction

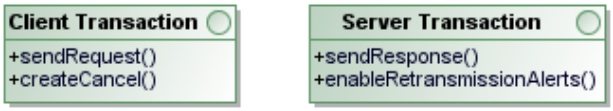

Figura 11. Diagrama de clase de Client y Server Transaction

Estos objetos se usan para enviar mensajes con estado. El Client Transaction se usa para las solicitudes, mientras que el Server Transaction se utiliza para las respuestas.

3.2.4 Cardinalidad

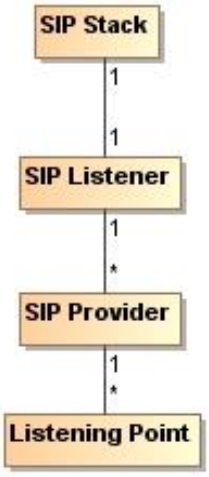

Figura 12. Diagrama de cardinalidad

La librería establece que cada SIP Stack debe tener un único SIP Listener y que sea un modelo de eventos unicast, es decir, que un SIP Provider solo puede tener registrado un SIP Listener. En cambio, un SIP Listener puede recibir eventos de varios SIP Provider.

Cada SIP Provider puede estar relacionado con cero o más Listening Point. Sin embargo, el SIP Provider solo puede tener un Listening Point para cada tipo de transporte. Las aplicaciones que deseen tener múltiples Listening Point con el mismo tipo de transporte deben utilizar un SIP Provider separado para cada Listening Point.

3.2.5 Configuración del SIP Stack

Para crear un SIP Stack, invocamos el método de SIP Factory "createSipStack (Properties)". Este método se configura utilizando un objeto java.util.Properties, En este objeto incluimos las propiedades necesarias para inicializar el SIP Stack [9].

Las propiedades generales de la API JAIN SIP que vamos a modificar (android.javax.sip):

STACK_NAME: Establece un nombre amigable para identificar la implementación de la pila. El nombre de la pila no debe contener espacios. Esta propiedad es obligatoria.

OUTBOUND_PROXY: Establece el proxy de salida de la pila SIP. El formato para esta cadena es "direcciónIP:puerto/transporte", por ejemplo, 129.1.22.333:5060/UDP. Esta propiedad es opcional.

Las propiedades específicas de JAIN SIP RI (android.gov.nist.javax.sip):

ENABLED_CIPHER_SUITES: Establece los conjuntos de cifrado de TLS permitidos.

TLS_CLIENT_AUTH_TYPE: Los valores válidos son: Default, Enabled, Want, Disabled o DisabledAll.

> Enabled: La pila SSL requerirá una cadena de certificados válida del cliente antes de aceptar una conexión.
>
> Want: La pila SSL solicitara un certificado, pero no falla si no se presenta uno
>
> Disabled: La pila SSL no requerirá una cadena de certificados para la conexión del servidor
>
> DisabledAll: La pila SSL no requerirá una cadena de certificados tanto para el servidor como para el cliente.

Adicionalmente hay que inicializar las propiedades de javax.net.ssl para establecer el certificado necesario para usar TLS.

keyStore: Por defecto es NULL. Si no se define, el keyStore y trustStore se dejarán con los valores predeterminados de la máquina virtual Java. Si se define, todos los sockets TLS creados utilizarán el almacén

de claves proporcionado.

keyStorePassword: Establece la contraseña que se usara para acceder al almacén de claves

trustStore: Si no se define, el trust store usara el mismo fichero que el keyStore.

trustStorePassword: Si no se proporciona una contraseña para el trustStore, se utilizará la contraseña del keyStore para ambos.

```
//Declaramos el router para los mensajes sip
if(proxyMode.equals("ON")) {
    properties.setProperty("android.javax.sip.OUTBOUND_PROXY", ipAddressProxy + ":" + portProxy + "/" + transportProtocol);
}
properties.setProperty("android.javax.sip.STACK_NAME", name);
properties.setProperty("android.gov.nist.javax.sip.ENABLED_CIPHER_SUITES","TLS_RSA_WITH_AES_128_CBC_SHA");
properties.setProperty("android.gov.nist.javax.sip.TLS_CLIENT_AUTH_TYPE", "DisabledAll");
```

Figura 13. Extracto del código de inicialización de las propiedades del SIP Stack en SCAIP Listener

3.3 Proxy ASTERISK

3.3.1 Análisis de soluciones

En el mercado hay una amplia variedad de bibliotecas de software libre para implementar un proxy SIP. Durante nuestro análisis, nos enfocamos en encontrar una solución que priorizara la facilidad de implementación y disponibilidad de documentación. De estas soluciones hemos decidido analizar las tres más usadas Asterisk, OpenSIPS y Kamilio.

Características	Asterisk	OpenSIPS	Kamilio
Tipo de software	Framework para construir aplicaciones de comunicaciones, incluyendo servidor SIP.	Servidor SIP	Servidor SIP
Primera version	Noviembre, 2005 1.2.1	Marzo, 2009 1.4.1	Septiembre, 2002 1.0
Ultima version	7 de Julio, 2023 20.3.1	21 de Junio, 2023 3.3.6	28 de Junio, 2023 5.7.1
Fortaleza	Amplia gama de funcionalidades. Mas amigable.	Enrutamiento de llamadas y soluciones VOIP a gran Escala. Altamente escalable	Enrutador y equilibrador de carga. Altamente escalable
Compatibilidad	Linux y otros sistemas Unix	Linux y otros sistemas Unix	Linux y otros sistemas Unix
Robustez	Robusto	Altamente robusto	Altamente robusto
Tipo de implementación	Software de uso libre	Software de uso libre	Software de uso libre
Documentación y recursos	Alto nivel de documentación	Alto nivel de documentación	Alto nivel de documentación
Comunidad y soporte	Comunidad y soporte activos	Comunidad y soporte activos	Comunidad y soporte activos
Seguridad	Autenticación de usuarios, cifrado de señalización (TLS/SSL)	Autenticación de usuarios, cifrado de señalización (TLS/SSL)	Autenticación de usuarios, cifrado de señalización (TLS/SSL)

Tabla 4. Análisis de soluciones de proxy SIP [10] [11] [12].

Después de analizar las tres soluciones, encontramos que todas son soluciones actuales y cumplen las características necesarias para nuestro caso. De estas soluciones, decidimos elegir Asterisk ya que, aunque no es tan escalable como las otras dos opciones, tiene la reputación de ser accesible y tener una menor curva de aprendizaje, especialmente para escenarios de prueba simples como el nuestro.

ASTERISK actuará como proxy SIP que permitirá el registro del cliente y el servidor SCAIP redirigiendo los mensajes entre ellos. Este programa de Linux es de código abierto y se ejecutará en una máquina virtual. Dicha máquina virtual estará conectada directamente (mediante un puente físico) a la red LAN en la que se desarrolla nuestro escenario. Utilizaremos Oracle VM para ejecutar la máquina virtual, utilizando la versión de Linux que la universidad nos proporciona. Dado que este aspecto es un complemento y no es el enfoque principal de

este trabajo, no profundizaremos demasiado en los detalles.

3.3.2 Introducción

Asterisk es un software de código abierto que proporciona una plataforma para construir sistemas de comunicaciones, especialmente centrados en VoIP. Asterisk ofrece funcionalidades de una central telefónica privada (PBX), así como capacidades avanzadas de enrutamiento de llamadas y comunicaciones en tiempo real. Además de la voz, Asterisk también es capaz de manejar otros medios de comunicación, como video y mensajería instantánea.

En Asterisk, los contextos y las extensiones son elementos clave en el enrutamiento y control de llamadas. Los contextos son agrupaciones lógicas que contienen reglas de marcado y configuraciones específicas, permitiendo organizar y segmentar la funcionalidad de Asterisk. Al asignar extensiones a contextos, se controla el acceso a recursos y se aplican políticas de enrutamiento. Por otro lado, las extensiones representan números internos asociados a acciones y aplicaciones, determinando cómo se manejan las llamadas. Con una configuración adecuada de contextos y extensiones, se crean flujos de llamadas personalizados y optimizados, garantizando una comunicación eficiente en Asterisk. Estos elementos se configuran en los archivos sip.conf y extensions.conf.

3.3.3 Instalación y configuración

Para simplificar el proceso, optaremos por instalar Asterisk a través de los repositorios utilizando el comando "apt-get install Asterisk". Esta instalación incluye una configuración general que permite iniciar Asterisk de inmediato y simplifica la gestión de la configuración. Como Asterisk es un servicio, se puede controlar utilizando el comando "service". La versión específica de Asterisk que utilizaremos es la 18.10.0. Para lograr la funcionalidad deseada en nuestro proyecto, realizaremos modificaciones en los archivos sip.conf y extensions.conf.

3.3.3.1 Sip.conf

Sip.conf es el archivo principal de configuración de Asterisk, donde se definen los campos relevantes que nos permiten personalizar y ajustar el funcionamiento del sistema. Esto incluye la configuración de usuarios. Los campos relevantes son:

Context: Especifica el contexto por defecto al que pertenece el usuario o dispositivo SIP.

Allowoverlap: Determina si se permite o no el solapamiento de dígitos en los números de marcado. Si se habilita, Asterisk permitirá números de marcado superpuestos.

Udpbindaddr: Define la dirección IP en la que Asterisk escucha las conexiones SIP UDP entrantes. Se utiliza para especificar la interfaz de red en la que se reciben las llamadas SIP UDP. Si es 0.0.0.0, escuchará en todas las interfaces.

Tcpenable: Indica si se habilita o no el soporte para conexiones SIP TCP. Si se establece en "yes", Asterisk aceptará conexiones TCP para las comunicaciones SIP.

Tcpbindaddr: Especifica la dirección IP en la que Asterisk escucha las conexiones SIP TCP entrantes. Se utiliza para definir la interfaz de red en la que se reciben las llamadas SIP TCP. Si es 0.0.0.0, escuchará en todas las interfaces.

Tlsenable: Determina si se habilita el soporte para conexiones SIP sobre TLS (Transport Layer Security). Si se configura en "yes", Asterisk aceptará conexiones seguras SIP sobre TLS.

Tlsbindaddr: Indica la dirección IP en la que Asterisk escucha las conexiones SIP sobre TLS entrantes. Se utiliza para especificar la interfaz de red en la que se reciben las llamadas SIP seguras.

Transport: Define el protocolo de transporte utilizado para las comunicaciones SIP. Puede ser UDP, TCP o TLS. El orden determina el protocolo por defecto.

Tlscertificate: Especifica la ruta al certificado utilizado para establecer conexiones SIP seguras sobre TLS. Debe apuntar al archivo de certificado correspondiente.

Qualify: Determina si Asterisk realiza un seguimiento del estado de los dispositivos SIP verificando su

disponibilidad. Si se habilita, Asterisk enviará paquetes de control periódicamente para monitorear la calidad y la disponibilidad del dispositivo.

Auth: Configura las opciones de autenticación para los usuarios SIP. Permite definir los mecanismos de autenticación utilizados, como el nombre de usuario y la contraseña.

Messages: Habilita o deshabilita el soporte para mensajes SIP entre usuarios. Si se establece en "yes", se permitirá el envío y recepción de mensajes SIP.

Sipdebug: Activa o desactiva el modo de depuración de SIP. Cuando está habilitado, Asterisk mostrará información detallada de depuración relacionada con las comunicaciones SIP en los registros de depuración.

Para crear nuestros usuarios, vamos a utilizar plantillas para establecer una configuración general que luego aplicaremos a usuarios individuales. Usaremos la notación [“nombreplantilla”](!) para definir una plantilla y la notación [“usuario”](nombreplantilla) para crear usuarios basados en esa plantilla. En la plantilla y en los usuarios los campos que se configuran son:

Type: Especifica el tipo de dispositivo de la extensión. Puede ser "friend", "user" o "peer". "Friend" indica que la extensión puede recibir y realizar llamadas, "user" indica que solo puede recibir llamadas y "peer" indica que solo puede realizar llamadas.

Host: Define la dirección IP o el nombre de dominio del dispositivo o servidor al que está asociada la extensión. Indica la ubicación donde se encuentra el dispositivo SIP asociado a la extensión.

Context: Determina el contexto al que pertenece la extensión. Define las reglas de enrutamiento y las acciones que se deben tomar cuando se reciben llamadas a esta extensión.

Username: Especifica el nombre de usuario asociado a la extensión. Es el identificador utilizado para autenticar y registrar la extensión en el servidor SIP.

Secret: Define la contraseña o clave secreta utilizada para autenticar la extensión. Es necesario proporcionar la contraseña correcta para que la extensión pueda realizar o recibir llamadas.

```
[general]
context=public                  ; Default context for incoming calls. Defaults to 'default'
allowoverlap=no                 ; Disable overlap dialing support. (Default is yes)
udpbindaddr=0.0.0.0             ; IP address to bind UDP listen socket to (0.0.0.0 binds to all)
tcpenable=yes                   ; Enable server for incoming TCP connections (default is no)
tcpbindaddr=0.0.0.0             ; IP address for TCP server to bind to (0.0.0.0 binds to all interfaces)
tlsenable=yes
tlsbindaddr=0.0.0.0
transport=udp,tcp, tls         ; Set the default transports.  The order determines the primary default transport.
srvlookup=yes                   ; Enable DNS SRV lookups on outbound calls
tlscertfile=/etc/asterisk/keys/asterisk.pem

qualify=yes
auth=none
messages=yes
```

Figura 14. Extracto de configuración del archivo sip.conf (1)

```
[clienteSCAIP](usuario)
username=clienteSCAIP
secret=s1234

[servidorSCAIP](usuario)
username=servidorSCAIP
secret=s1234
```

```
[usuario](!)
type=friend
host=dynamic
context=scaip
```

Figura 15. Extracto de configuración del archivo sip.conf (2)

3.3.3.2 Extension.conf

En este archivo vamos a escribir las reglas del contexto cuando los usuarios maquen la extensión para que consigamos las funcionalidades deseadas. En este proxy vamos a configurar dos tipos distintos de funcionalidad, uno de ellos en el que en mensaje que recibamos sea enviado a la extensión marcada y el segundo en el que el mensaje sea enviado a la dirección SIP marcada.

Las dos reglas del contexto para la primera funcionalidad serian:

Exten => _.,1,NoOp(SIP Message received: $[{MESSAGE(body)})

Same => n,MessageSend(sip:${EXTEN}, ${MESSAGE(from)})

La primera línea define que tras escribir cualquier extensión (el guion bajo representa cualquier extensión) y se imprime un mensaje de registro en el archivo de registro de Asterisk. En este caso, se muestra el mensaje "SIP Message received: " seguido del contenido del cuerpo del mensaje SIP recibido.

La segunda línea continua a la primera (la letra "n" indica "next") y se utiliza para enviar un mensaje SIP a una dirección específica. En este caso, el mensaje se envía al valor de la variable ${EXTEN}, que representa la extensión actual, y ${MESSAGE(from)} que representa la dirección del remitente del mensaje SIP recibido.

Para implementar la segunda funcionalidad tendríamos que cambiar la segunda línea por "Same => n,MessageSend(${MESSAGE(to)}, ${MESSAGE(from)})". Este cambio haría que el mensaje se envíe a la dirección especificada en la variable ${MESSAGE(to)}, que representa el destino del mensaje SIP recibido.

```
[scaip]
exten => _.,1,NoOp(SIP Message received: ${MESSAGE(body)})
same => n,MessageSend(sip:${EXTEN},${MESSAGE(from)})
;same => n,MessageSend(${MESSAGE(to)},${MESSAGE(from)})
```

Figura 16. Extracto de configuración del archivo extensions.conf

3.3.3.3 Certificado TLS

Para obtener los certificados necesarios para la comunicación, se requiere utilizar un script que se encuentra en la distribución de ASTERISK. Para ello, se puede descargar la versión 18 del software con el comando wget http://downloads.asterisk.org/pub/telephony/asterisk/asterisk-18-current.tar.gz. Luego, se descomprime el archivo descargado con el comando tar -zxvf asterisk-18-current.tar.gz.

Una vez descomprimido, se accede a la carpeta contrib/scripts, donde se encontrará el script necesario para crear el certificado llamado "ast_tls_cert". Para crearlo, se ejecuta el comando "./ast_tls_cert -C "Asterisk Private CA" -O "SCAIProxy" -d /etc/Asterisk/keys". Es importante asegurarse de otorgar los permisos adecuados para que el usuario que ejecute Asterisk, en este caso "asterisk", pueda leer estos certificados. Esto se podrá conseguir con el comando chmod.

4 PRACTICA

En esta sección de nuestro trabajo, vamos a examinar las aplicaciones desarrolladas en Android Studio y su arquitectura. Nuestra aplicación principal es SClock, un emisor de alarmas SCAIP. Además, hemos creado una aplicación secundaria que actúa como servidor de alarmas SCAIP que nos permite tener un flujo completo de mensajes para en la sección de pruebas, SServer. Las aplicaciones solo van a desarrollar el primer caso de uso de la especificación SCAIP, el intercambio de alarmas sin intercambio de voz o datos.

El código de todas las aplicaciones se ha subido a GitHub para proporcionar una mayor claridad. Este código puede accederse a través del siguiente enlace: https://github.com/GermanBlas/SCAIP-TFG.

4.1 Arquitectura Cliente: SClock

SClock es una aplicación móvil desarrollada en Java utilizando Android Studio. Su función principal es registrarse en un proxy SIP y enviar mensajes SCAIP de dos formas distintas. La primera forma es manual, donde los usuarios interactúan en la aplicación para enviar mensajes SCAIP a través de la red. La segunda opción permite programar alarmas para que se envíen periódicamente.

La arquitectura de SClock se divide en 3 paquetes. El paquete SCAIP contiene las clases encargadas de la funcionalidad de mensajería del protocolo SCAIP y SIP. El paquete actividades encapsula las actividades que representan las pantallas con las que los usuarios pueden interactuar. Por último, el paquete de la base de datos incluye las clases necesarias para gestionar la base de datos de la aplicación. Además, contaremos con un paquete de recursos donde se almacenarán los archivos necesarios para la funcionalidad completa de la aplicación.

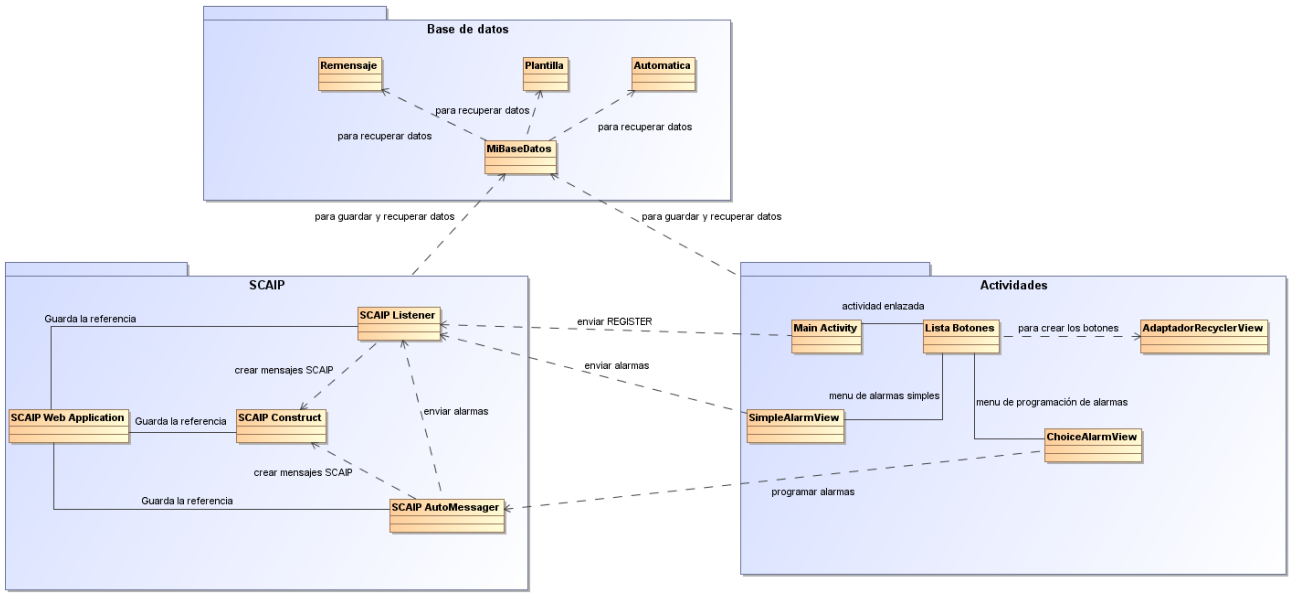

Figura 17. Arquitectura de clases de SClock

4.1.1 Paquete SCAIP

4.1.1.1 SCAIP Listener

Figura 18. Diagrama de clase de SCAIP Listener

La clase central de nuestra aplicación Android, responsable de brindar las funcionalidades de mensajería. Se trata de una encapsulación de JAIN SIP RI con la adición de la funcionalidad de SCAIP.

La función "init" se utiliza para inicializar la clase mediante la configuración de diferentes objetos SIP. Los argumentos que se pasan son los siguientes:

Name, ipAddress, port: Estos parámetros especifican el nombre, dirección IP y puerto del SIP Listener.

IpAddressProxy y PortProxy: Estos parámetros especifican la dirección IP y puertos del proxy que utilizaremos como gateway.

TransportProtocol: Este parámetro indica el protocolo de transporte que vamos a utilizar.

ProxyMode: Este parámetro indica si vamos a utilizar un proxy o no. Si está activado ("ON"), todos los paquetes se enviarán al proxy. Si está desactivado, se ignorarán los parámetros del proxy y los paquetes se enviarán directamente a la dirección IP de las peticiones.

En primer lugar, inicializamos la SIP Factory con el nombre de ruta (pathname) de la implementación de referencia (android.gov.nist). El siguiente paso es inicializar el SIP Stack, lo cual se logra configurando las propiedades necesarias para su configuración. A continuación, creamos un Listening Point, que representa el socket de escucha de este listener. También creamos un SIP Provider, responsable de generar los eventos que se enviarán al Sip Listener al que se registra, en este caso, nuestro propio Listener.

Posteriormente, inicializamos las otras factorías necesarias en los métodos subsiguientes.

Los siguientes métodos se encargan de crear mensajes de petición SIP. Los argumentos necesarios son: toSipAddress (IP del destinatario), toUser (nombre SIP del destinatario), toPort (puerto del destinatario) y requestMethod (método de solicitud SIP). Si se proporciona un quinto argumento, dataMessage, se incluirá como contenido en el mensaje SIP, lo cual utilizaremos para encapsular el mensaje SCAIP dentro del mensaje

SIP.

Estos métodos crean las diversas cabeceras requeridas en el mensaje SIP y generan el objeto de petición SIP que se enviará posteriormente en su respectivo método. Las variables necesarias para crear estas cabeceras se obtienen de los argumentos proporcionados o de las inicializaciones previas, como la IP de origen del mensaje. Además de estos métodos, los métodos “createRequestProxy” y “createRequestEnd” son versiones simplificadas que crean una solicitud al servidor o al punto final al incluir solo los argumentos message (contenido del mensaje) y method (método de la solicitud).

El método "sendRequest" se encarga de enviar la solicitud (Request) pasada como argumento a través del SIP Provider.

A continuación, encontramos métodos que se centran en procesar los diferentes tipos de eventos que el SIP Provider puede pasar al SIP Listener.

El método "processRequest" analiza el evento de solicitud (Request) recibido y, dependiendo del método de la solicitud, lo redirige a una función específica que procesa esa solicitud SIP en particular. En este escenario, solo utilizaremos los métodos "processMessage" y "processOptions".

El método "processMessage" se encarga de procesar los mensajes recibidos. Cuando se recibe un mensaje, construimos una respuesta "200 OK" para informar que se ha recibido correctamente, y la enviamos al mismo destino desde donde se recibió. Si este mensaje es una solicitud (Request), la aplicación responderá con una respuesta (Response) SCAIP exitosa.

El método "processOptions" se utiliza para responder a los mensajes OPTIONS recibidos por el servidor. Cuando nos registramos en el servidor, este verifica nuestra disponibilidad y solicita información sobre nuestras capacidades mediante una solicitud OPTIONS. En nuestro caso, no necesitamos informar al servidor sobre nuestras capacidades, ya que solo nos comunicaremos a través de mensajes MESSAGE. Por lo tanto, solo debemos responder con un código "200 OK".

El método "processResponse" se utiliza para analizar las líneas de respuesta de los mensajes de respuesta SIP. Principalmente, se utiliza para proporcionar funcionalidad de autenticación a la aplicación. Cuando enviamos un mensaje al servidor, este nos desafía con una cabecera de autenticación. Cuando este método detecta el desafío, reenvía el mensaje con la cabecera de autenticación completada. Para calcular la respuesta de autenticación en SIP, se utiliza una fórmula específica la cual requiere seis elementos.

HA1=MD5(username:realm:password)

HA2=MD5(method:digestURI)

response=MD5(HA1:nonce:HA2)

El **realm** y el **nonce** se obtienen de la cabecera de autenticación. La **URI** se obtiene del ToHeader y el **método** del CSeqHeader. El nombre de usuario (**username**) y la contraseña (**password**) son elementos que debemos proporcionar nosotros mismos. Una vez que tenemos esta información, utilizamos la fórmula para crear la respuesta que ira dentro de la cabecera de autenticación. Para reconstruir el mensaje enviado, se recuperará la información necesaria de la base de datos. Esta funcionalidad se explica en más detalle en la sección de la base de datos. Además de incluir la cabecera de autenticación, debemos incrementar el valor del campo CSeq del mensaje que vamos a reenviar.

Los otros cuatro métodos quedan sin definir para una posible ampliación del trabajo.

4.1.1.2 SCAIP Construct

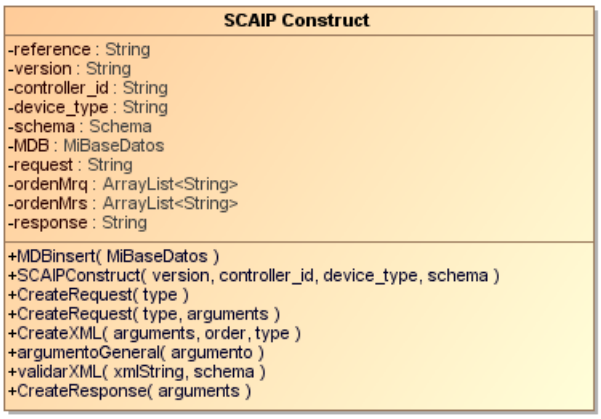

Figura 19. Diagrama de clase de SCAIP Construct

La principal función de esta clase es construir los mensajes SCAIP que serán encapsulados en SIP. Los campos de los mensajes SCAIP se dividen en tres tipos: generales para todos los mensajes, específicos para cada plantilla de mensajes y únicos para cada mensaje. Los argumentos generales se incluyen en el constructor de la clase, junto con el esquema XSD (XML Schema Definition) del protocolo SCAIP. Al utilizar el método de creación del mensaje SCAIP, se pueden proporcionar los argumentos únicos a través de un HashMap.

Un esquema XSD es una especificación que define la estructura y los tipos de datos permitidos en un documento XML, proporcionando un conjunto de reglas y restricciones que se pueden utilizar para validar y verificar la integridad de los documentos XML[13].

Durante la construcción del mensaje SCAIP, se recupera la plantilla correspondiente, que contiene los argumentos específicos de la plantilla. Con todos los argumentos necesarios, se construye el mensaje SCAIP respetando el orden de los elementos XML utilizando un ArrayList que determina el orden de cada uno de ellos. Una vez construido el mensaje, se valida con el esquema XSD del SCAIP. Si la validación es exitosa, se devuelve el mensaje.

4.1.1.3 SCAIP AutoMessager

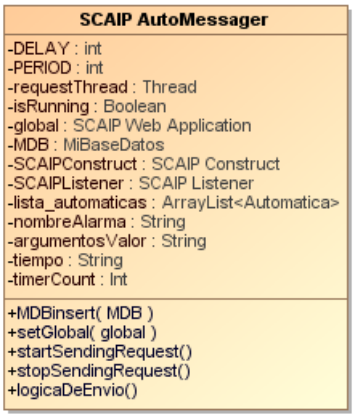

Figura 20. Diagrama de clase de SCAIP AutoMessager

La funcionalidad de programar alarmas y enviarlas periódicamente se implementa a través de la clase AutoMessager. Esta clase crea un hilo que se encarga de enviar las alarmas programadas de manera constante. El hilo está configurado con un tiempo de delay, que permite esperar un período determinado antes de iniciar el hilo, para asegurarse de que la aplicación esté completamente configurada, y un tiempo de periodo, que activa la funcionalidad de envío de mensajes a intervalos regulares.

El objetivo del hilo es, cuando se alcanza el tiempo de periodo, recuperar de la base de datos una lista de alarmas programadas. Esta lista contiene información sobre cuándo debe ser enviada cada alarma, basándose

en los periodos transcurridos. Si una alarma debe ser enviada en ese momento, se ejecuta la lógica de envío correspondiente.

La lógica de envío implica construir el mensaje SCAIP utilizando los argumentos almacenados en la base de datos, y luego encapsularlo en un mensaje SIP que se guarda en caso de que sea necesario reenviarlo debido a autenticación.

Para iniciar y parar el hilo hacemos uso de las funciones “startSendingRequests” y “stopSendingRequest”

4.1.2 SCAIP Web Application

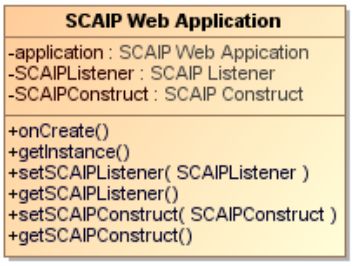

Figura 21. Diagrama de clase de SCAIP Web Application

La clase "SCAIP Web Application" es una subclase de la clase "Application" en Android. La clase “Application” es una clase base especial que se utiliza para representar la instancia global de la aplicación y se utiliza para mantener el estado global y las configuraciones de la aplicación durante todo su ciclo de vida. Vamos a usar esta clase para guardar las instancias de objetos para compartirlas en los demás puntos de la aplicación. En particular, vamos a almacenar las instancias del SCAIP Listener y el SCAIP Construct en esta clase.

4.1.3 Paquete Actividades

4.1.3.1 Main Activity

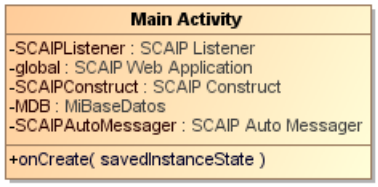

Figura 22. Diagrama de clase SCAIP de Main Activity

La vista Main Activity es la primera actividad que se ejecuta al iniciar la aplicación. Esta actividad se encarga de inicializar los diferentes módulos que se utilizan en la aplicación. Mientras, muestra el logo de la aplicación.

En primer lugar, se obtiene la instancia de la aplicación SCAIP Web Application (se llamará global) y se inicializa la base de datos. A continuación, se carga en la memoria del dispositivo el esquema XSD de SCAIP para inicializar SCAIP Construct, guardando el objeto en global. Posteriormente, se cargan los certificados de los recursos y se inicializan en las propiedades del sistema.

Luego, se inicializa el módulo SCAIP Listener con los datos proporcionados, guardando la configuración en la aplicación global. Finalmente, se inicializa el módulo SCAIP AutoMessager.

Después de inicializar todos los módulos necesarios, se crea una petición REQUEST para registrarse en el proxy. Dado que no se pueden enviar mensajes en el hilo principal, se crea un hilo adicional para enviar la solicitud.

Una vez completado este proceso, se pasa a la actividad principal de la aplicación, que es la lista de botones.

4.1.3.2 Lista Botones

Figura 23. Diagrama de clase de Lista Botones

Esta actividad será el menú principal, donde se muestran todas las alarmas que pueden ser enviadas. El primer paso es cargar las plantillas en la base de datos, utilizando el método insertarPlantillas() de la clase. Este método ejecuta instrucciones de base de datos para incluir las plantillas correspondientes. Los diferentes tipos de alarmas y sus respuestas están definidos en el apéndice de la memoria.

La vista de la lista se crea dinámicamente utilizando un RecyclerView. Un RecyclerView es una vista flexible que permite mostrar una lista de elementos desplazable. A diferencia de una vista de lista convencional, el RecyclerView optimiza el uso de recursos al eliminar y crear elementos dinámicamente mientras se desplaza por la pantalla. Para cargar los datos en el RecyclerView, se utiliza un adaptador. El adaptador se encarga de convertir una colección de datos en elementos visibles y se enlaza con el RecyclerView. En esta vista, se inicializa el adaptador del RecyclerView y se le pasan los nombres de las plantillas. Además, se sobrescribe el método onEntrada para que, al pulsar un botón, se active el método GoAlarm con el nombre de ese botón.

El método GoAlarm nos lleva al menú de envío de alarmas. Las alarmas se dividen en dos grupos: manuales y automáticas. Dependiendo de esto, se nos redirige a la vista Simple Alarm View (alarma manual) o Choice Alarm View (alarma automática). El nombre de la alarma pulsada se envía dentro del Intent al menú.

4.1.3.3 Adaptador RecyclerView

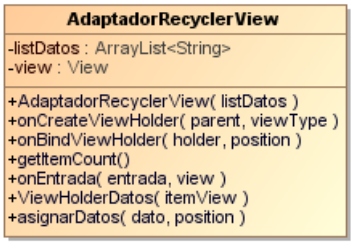

Figura 24. Diagrama de clase de Adaptador RecyclerView

Esta clase abstracta se encarga de configurar el adaptador del RecyclerView para que, por cada dato que reciba (en este caso, el nombre de todas las posibles alarmas), construya un botón al que se le asigna dicho nombre y se ejecute el método onEntrada. En la clase ListaBotones, hemos sobrescrito el método onEntrada para que, cuando se pulse un botón, invoque el método GoAlarm con el nombre de la alarma correspondiente.

4.1.3.4 Simple Alarm View

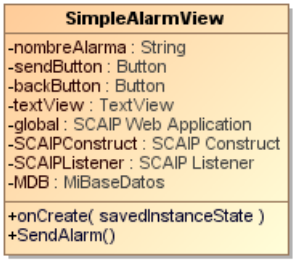

Figura 25. Diagrama de clase de Simple Alarm View

Este menú está destinado a las alarmas manuales, que son aquellas que enviamos únicamente a través de la interacción con el usuario. Al ingresar a este menú, se muestra el tipo de alarma que se enviará y se presentan

dos botones. Uno de ellos nos devuelve a la actividad anterior y el otro se encarga de enviar la alarma. Al enviar la alarma, se ejecuta el método SendAlarm(). En este método, se construye el mensaje SCAIP y luego se encapsula en un mensaje SIP utilizando el método createRequestServer(). Una vez construido, el mensaje se guarda en la base de datos por si es necesario reenviarlo debido a la autenticación, y se crea un hilo para enviarlo. Después de esto, se regresa a la vista principal, Lista Botones.

4.1.3.5 Choice Alarm View

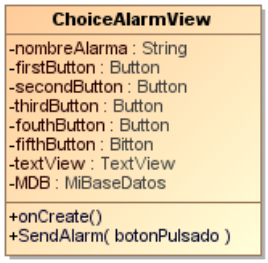

Figura 26. Diagrama de clase de Choice Alarm View

Este menú está diseñado para configurar alarmas automáticas, que son aquellas que programamos para que sean enviadas periódicamente por el Auto Messager. Al ingresar a este menú, se muestra el tipo de alarma que se está configurando, un botón para regresar y cinco botones que nos permiten seleccionar el comportamiento de esta alarma. Los botones numerados indican la frecuencia de envío de la alarma en términos de períodos. Si se pulsa alguno de estos botones, se creará una entrada en la tabla "Automaticas" para programar el envío de la alarma. Por otro lado, si se selecciona el botón de cancelación, se cancelará la programación de envío de las alarmas automáticas, en caso de que estuviera programada previamente. Después de realizar la configuración, la aplicación nos devolverá a la vista principal, "Lista Botones".

4.1.4 Paquete Base de datos

Este paquete contiene todas las clases necesarias para el funcionamiento de nuestra base de datos SQLite.

4.1.4.1 MiBaseDatos

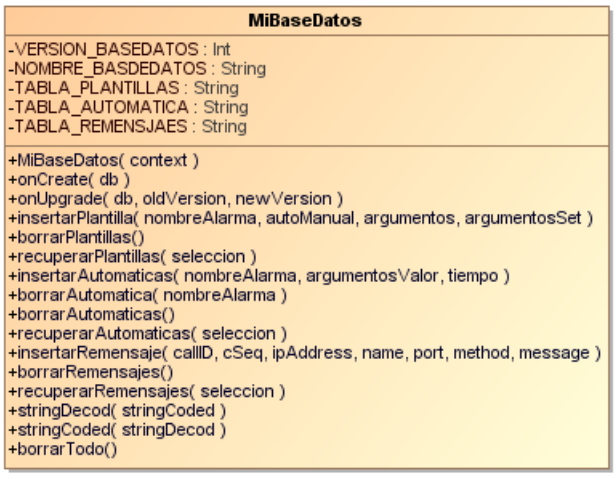

Figura 27. Diagrama de clase de MiBaseDatos

Esta clase se encarga de gestionar la dinámica de ejecución de las bases de datos utilizadas en la aplicación. En este caso, se tienen tres bases de datos: "Plantillas", "Automaticas" y "ReMensajes", que serán analizadas posteriormente.

La estructura de esta clase consiste en la definición de las tablas y sus respectivos campos, así como la declaración de las claves primarias. También se incluyen métodos para la creación de estas tablas, así como métodos específicos para cada base de datos que permiten la inserción, eliminación y recuperación de tuplas en las mismas.

4.1.4.2 Plantillas

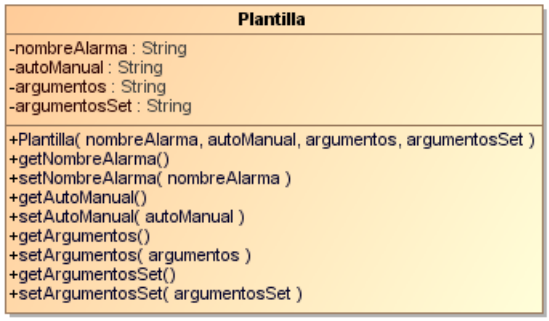

Figura 28. Diagrama de clase de Plantilla

La base de datos "Plantillas" se utiliza para almacenar las plantillas que se utilizarán para construir los archivos XML de SCAIP. La tabla de esta base de datos tiene los siguientes campos:

"nombreAlarma": Indica el nombre del tipo de alarma.

"autoManual": Es una variable que indica si la alarma está diseñada para ser enviada manualmente o para ser programada y enviada por el AutoMessager.

"argumentos": Guarda los argumentos XML que deben ser completados para construir el mensaje SCAIP. Estos argumentos están separados por el carácter "::".

"argumentosSet": Guarda los argumentos específicos de esa alarma, que también están separados por el carácter "::" y se presentan en la notación "argumento=valor".

La clave primaria de la tabla es "nombreAlarma", lo que significa que solo puede haber una plantilla por cada tipo de alarma.

El objeto JSON utilizado para interactuar con esta base de datos es "Plantilla".

4.1.4.3 Automaticas

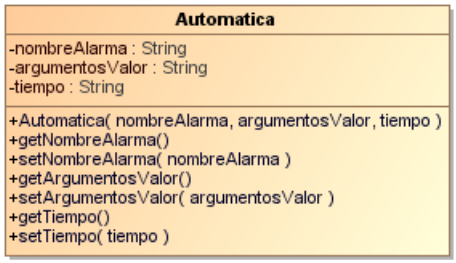

Figura 29. Diagrama de clase de Automatica

La base de datos "Automaticas" se utiliza para almacenar los mensajes SCAIP programados que deben ser enviados periódicamente por el AutoMessager. La tabla de esta base de datos tiene los siguientes campos:

"nombreAlarma": Indica el nombre del tipo de alarma.

"argumentosValor": Guarda los valores únicos de la alarma que se reenviará. Estos argumentos están separados por el carácter "::" y se presentan en la notación "argumento=valor".

"tiempo": Indica el período de tiempo en el que se debe enviar la alarma.

La clave primaria de la tabla es "nombreAlarma", lo que significa que solo puede haber una entrada por cada tipo de alarma.

El objeto JSON utilizado para interactuar con esta base de datos es "Automatica".

4.1.4.4 Remensajes

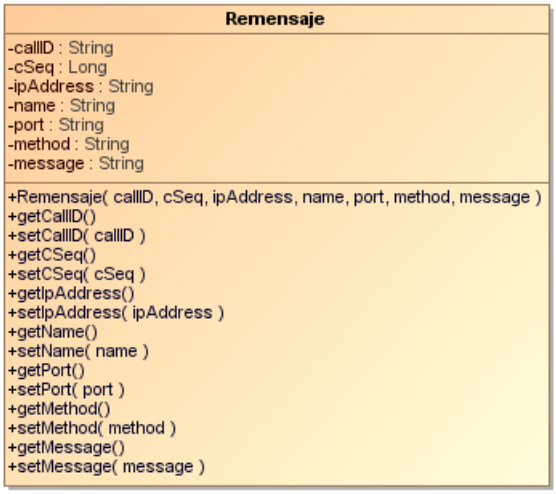

Figura 30. Diagrama de clase Remensaje

La base de datos "Remensajes" se utiliza para almacenar la información necesaria para reenviar mensajes en el caso de que sea necesario, especialmente cuando se produce un desafío de autenticación Digest por parte del servidor. La tabla de esta base de datos tiene los siguientes campos:

"callID": Almacena el campo "callID" del mensaje, el cual es único y se utilizará para identificar el mensaje.

"cSeq": Guarda el número de secuencia del mensaje.

"ipAddress", "name", "port": Almacenan la dirección IP, el nombre y el puerto a los que se dirige la petición.

"method": Guarda el método de la petición.

"message": Almacena el contenido completo de la petición.

Estos argumentos se utilizan para reconstruir el mensaje y enviarlo nuevamente en el caso de que sea necesario debido a un desafío de autenticación.

El objeto JSON utilizado para interactuar con esta base de datos es "Remensaje".

4.1.5 Paquete de recursos.

En este paquete vamos a guardar distintos recursos necesarios para la aplicación.

4.1.5.1 Drawable

En esta carpeta vamos a guardar los distintos archivos CSS que vamos a usar para definir las propiedades visuales de los distintos objetos, como los botones de la lista de alarmas

4.1.5.2 Layout

En esta carpeta se encuentran los diferentes diseños necesarios para la aplicación, como la vista principal y otras vistas relacionadas. Los layouts específicos que se encuentran aquí son los siguientes:

activity_main.xml: Este layout corresponde a la actividad principal (MainActivity) y contiene únicamente un logo que se muestra mientras la aplicación se inicializa.

lista_layout.xml: Este layout se utiliza en la actividad Lista Botones y representa el marco del RecyclerView. El contenido del RecyclerView se carga dinámicamente utilizando un adaptador en tiempo de ejecución.

lista_layout_button.xml: Este layout define la plantilla de los botones que forman parte del RecyclerView en lista_layout.

choice_alarm_layout.xml: Esta vista muestra el nombre de la alarma que se está configurando, así como cinco

botones para establecer las opciones de la alarma. También incluye un botón para regresar a la actividad principal.

simple_alarm_layout.xml: Esta vista muestra el nombre de la alarma que será enviada, un botón para enviarla y un botón para regresar a la actividad principal.

4.1.5.3 Raw

Dentro de esta carpeta se encuentran los certificados necesarios para cifrar la comunicación mediante TLS. Estos certificados se generarán utilizando OpenSSL. A continuación, se muestran los comandos necesarios para crear el certificado de SClock:

Crear el certificado de SClock: "openssl req -x509 -out certificado.crt -keyout clave_privada.key -newkey rsa:2048 -days 1024 -nodes -sha256 -subj "/C=ES/O=ETSI/CN=SClock" -extensions EXT -config config.txt"

Convertir el certificado a formato PKCS12: "openssl pkcs12 -export -in certificado.crt -inkey clave_privada.key -out keystore.p12 -name SCAIPPa"

Importar el certificado PKCS12 a un keystore JKS: "keytool -importkeystore -destkeystore keystore.jks -srckeystore keystore.p12 -srcstoretype PKCS12"

Importar el certificado en un truststore JKS: "keytool -import -trustcacerts -alias SCAIPPa -file certificado.crt -keystore truststore.jks"

Es importante tener en cuenta que los certificados generados tendrán la extensión JKS. Sin embargo, para poder ser leídos por Android Studio, es necesario que tengan la extensión BKS. Esto se puede lograr utilizando un programa de administración de certificados, como KeyStore Explorer.

Además, dentro de esta carpeta también se encuentra el esquema XSD que se utilizará para la verificación de los mensajes SCAIP generados.

4.1.6 Manifiesto Android

El Manifiesto de Android es un archivo XML fundamental en el desarrollo de aplicaciones para la plataforma Android. Proporciona información sobre la aplicación como su nombre, version, los permisos requeridos y actividades.

Permisos requeridos:

WAKE_LOCK: Permite que la aplicación mantenga el dispositivo despierto y evita que la pantalla y el procesador se apaguen automáticamente.

INTERNET: Permite que la aplicación acceda a Internet.

ACCESS_NETWORK_STATE: Permite que la aplicación acceda al estado de la red.

ACCESS_WIFI_STATE: Permite que la aplicación acceda al estado del Wi-Fi.

READ_EXTERNAL_STORAGE: Permite que la aplicación lea el contenido almacenado en el almacenamiento externo del dispositivo. Esto se utiliza para leer los certificados y el esquema.

WRITE_EXTERNAL_STORAGE: Permite que la aplicación escriba en el almacenamiento externo del dispositivo. Es necesario para escribir los certificados y el esquema en la memoria.

Especificación de la aplicación global:

Para utilizar SCAIP Web Application, debemos especificar dentro de la etiqueta application su nombre mediante la línea android:name.

Actividades:

Especificamos las actividades que pueden ejecutarse en la aplicación. En este caso, se incluyen las 4 actividades de la aplicación.

Se marca la actividad que se ejecutará en el inicio como MainActivity.

4.1.7 Build.gradle

El archivo build.gradle es un archivo de configuración utilizado en proyectos desarrollados con el sistema de construcción Gradle. Contiene la configuración del proyecto y define cómo se compila, construye y empaqueta la aplicación. Específicamente, el archivo build.gradle define las dependencias, las tareas de compilación y los plugins que se utilizarán en el proyecto. Las dependencias necesarias en este trabajo son:

JAIN SIP RI: Esta es la biblioteca principal de la aplicación que proporciona funcionalidad para implementar una pila SIP.

Log4j y Android logging log4j: Estas bibliotecas de logging son necesarias para JAIN SIP.

Play-services-wearable: Esta biblioteca es parte del conjunto de servicios de Google Play y ofrece funcionalidad para dispositivos wearables. Viene por defecto.

Percentlayout: Esta biblioteca permite definir diseños de manera proporcional basándose en porcentajes, lo que facilita la creación de interfaces de usuario responsivas. Viene por defecto.

Legacy-support-v4: Esta biblioteca ayuda a garantizar que las aplicaciones sean compatibles con versiones antiguas de Android sin perder características y mejoras más recientes. Viene por defecto.

Recyclerview: Esta biblioteca de AndroidX proporciona la implementación de RecyclerView, que es un widget para mostrar listas de elementos desplazables y optimizadas en Android.

Androidx.wear: Esta biblioteca de AndroidX ofrece componentes y herramientas adicionales para el desarrollo de aplicaciones para dispositivos wearables con Wear OS. Viene por defecto.

Xerces:xercesImp: Esta es una biblioteca de Apache Xerces, que es una implementación de código abierto del estándar XML. Permite el procesamiento y análisis de documentos XML en aplicaciones. Es necesario para el manejo de SCAIP.

Org.bouncycastle:bcprov-jdk15on: Esta es una biblioteca de Bouncy Castle, que es una implementación de criptografía en Java. Proporciona funcionalidades de cifrado, firmas digitales y otros algoritmos criptográficos. Es necesario para el cifrado TLS.

Commons-cocec: Esta biblioteca de Apache Commons Codec proporciona utilidades para la codificación y decodificación de datos, como codificación base64, codificación URL y hash de datos. Ofrece funciones de cifrado MD5.

4.2 Arquitectura Servidor: SServer

Hemos desarrollado una aplicación complementaria llamada SServer, diseñada para funcionar como servidor de mensajes SCAIP. Esta aplicación utiliza las clases existentes para implementar la funcionalidad del servidor. Dado que el aspecto de SCAIP del servidor está fuera del alcance de nuestro estudio, se asumirá que todas las solicitudes SCAIP recibidas están correctamente formuladas y que pueden recibir una respuesta estándar de confirmación.

4.2.1 Paquete SCAIP

Dentro de este paquete solo encontraremos las clases SCAIP Construct y SCAIP Listener. Esto se debe a que no utilizaremos la funcionalidad del AutoMessager, ya que no necesitamos programar el envío de alarmas automáticas.

4.2.2 Paquete Actividades

La única actividad disponible en esta aplicación será la clase Main Activity. Esta clase se comportará de manera similar a la clase principal de la aplicación, inicializando los diversos módulos que la aplicación utilizará y registrándose en el servidor. Después de eso, permanecerá en modo pasivo hasta que lleguen las solicitudes SCAIP, a las cuales responderá a todas con una respuesta de éxito estándar.

4.2.3 Paquete Base de datos

Dentro del paquete de base de datos solo encontraremos dos de las tres bases de datos de la aplicación principal: Plantillas y Remensajes. La base de datos "Automaticas" ha sido eliminada, ya que no se implementará el módulo AutoMessager.

4.2.4 Android Manifest

El archivo Android Manifest es similar al anterior, con la única diferencia de que ahora solo se declara una actividad: Main Activity.

4.2.5 Build.gradle

El archivo build.gradle se mantiene igual.

4.3 Aplicación de Pruebas SCAIPPA

Durante el desarrollo de nuestro proyecto, creamos una aplicación de prueba en Android con la capacidad de modificar los valores principales del Listener y de los mensajes enviados. Esta aplicación se diseñó con el propósito de depurar y evaluar la funcionalidad del envío de mensajes. Aunque esta aplicación no se utiliza en el contexto de nuestro trabajo y no la analizaremos, hemos subido su código fuente a GitHub junto con las otras dos aplicaciones principales del proyecto. Esto permitirá que cualquier persona interesada en avanzar en este trabajo pueda utilizarla como referencia adicional.

5 PRUEBAS

En esta sección, vamos a presentar pruebas que demuestran el correcto funcionamiento de la aplicación, además de examinar detalladamente su funcionamiento secuencial y la interacción entre los diferentes objetos.

Como mencionamos anteriormente, nuestra implementación se centrará únicamente en el caso de uso de envío de mensajes sin intercambio de datos y voz. La configuración del proxy permitirá únicamente la redirección de mensajes a través de los nombres en las direcciones SIP.

Para facilitar el análisis de los mensajes, realizaremos las pruebas utilizando el protocolo TCP, lo que permitirá examinar los mensajes a través de Wireshark. Además, realizaremos una prueba adicional para comprobar el correcto funcionamiento en TLS.

5.1 Registro

En este apartado de pruebas, se describe la secuencia de inicio de ambas aplicaciones hasta el registro en el proxy Asterisk.

Al iniciar la aplicación, se muestra el logo de la aplicación en la pantalla mientras se van inicializando los diferentes módulos necesarios en cada aplicación. En el caso de la aplicación servidor SServer, solo se inicializarán los módulos SCAIP Construct y SCAIP Listener. Todas las referencias de los objetos SCAIP se guardarán en SCAIP Web Application.

Antes de crear el módulo SCAIP, es necesario cargar en memoria el esquema XDS. Luego, se procede a inicializar el módulo SCAIP Construct. Para ello, se pasan los argumentos necesarios en el constructor, como el esquema XDS y los argumentos generales para construir los mensajes SCAIP.

Antes de inicializar el Listener, se cargan en memoria los certificados. A continuación, se inicializa el SCAIP Listener. Durante esta inicialización, se crean las diferentes clases necesarias para la funcionalidad de mensajería SIP. La SIP Factory se encarga de construir el SIP Stack y las clases Address, Header y Message Factory. Utilizando el SIP Stack, se crea el Listening Point y el SIP Provider, que se configura para generar eventos cuando se reciben mensajes a través del Listening Point.

Los valores de inicialización del Listener son los siguientes:

name	IpAddress	Port	ipAddressProxy	portProxy	transportProtocol	proxyMode	nameEnd	ipAddressEnd	portEnd
clienteSCAIP	10.0.2.16	5060	192.168.1.251	5060	TCP	ON	servidorSCAIP	192.168.1.250	5060

Tabla 5. Valores de inicialización del SIP Listener de SClock

Para la aplicación servidor, el valor de "name" se intercambiarán con "nameEnd" y se tendrá que cambiar el valor de “ipAddressEnd” a 192.168.1.191. Es importante destacar que la configuración que utilizaremos en el proxy para las pruebas redirigirá los mensajes a través del nombre en la URI de destino, por lo que "ipAddressEnd" y "portEnd" no son necesarios, ya que el proxy obtendrá esta información de la base de datos por el registro de las aplicaciones.

Finalmente se inicializa el SCAIP Auto Messager

Figura 31. Pantallas de inicio de ambas aplicaciones

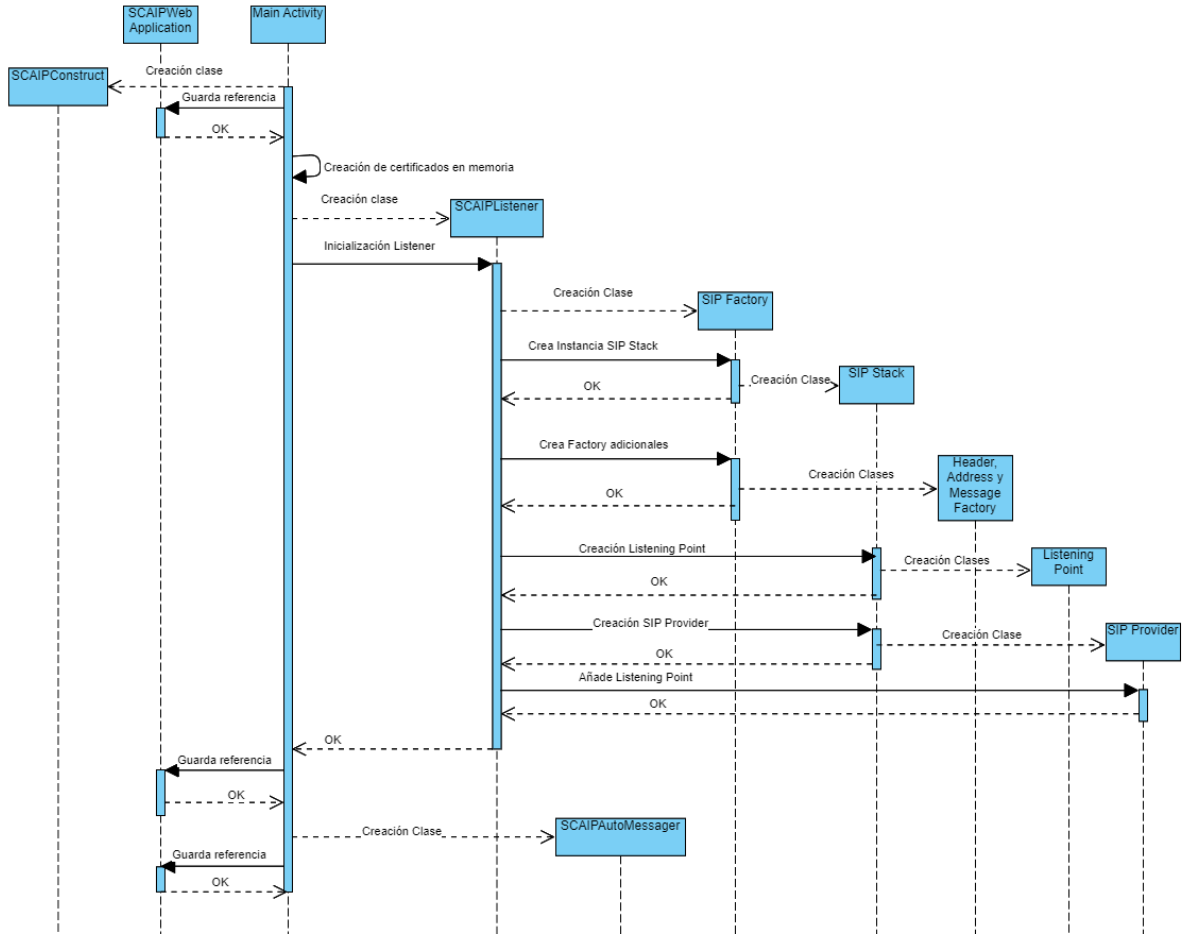

Figura 32. Diagrama de secuencia de la iniciación de módulos de la aplicación

A partir de este párrafo, vamos a analizar el proceso de registro de las aplicaciones en el proxy. En la tabla de intercambio de mensajes se muestran los valores más relevantes en esta interacción: la dirección IP y el puerto, IPS (IP Source) e IPD (IP Destination), los campos "From" y "To". Cabe destacar que estos campos no se intercambian al enviar las respuestas, así como la cabecera de autenticación.

Como mencionamos anteriormente, para registrarse en el proxy, en el mensaje de petición de registro (REGISTER), el campo "To" debe contener la dirección de usuario con la que deseamos establecer la conexión.

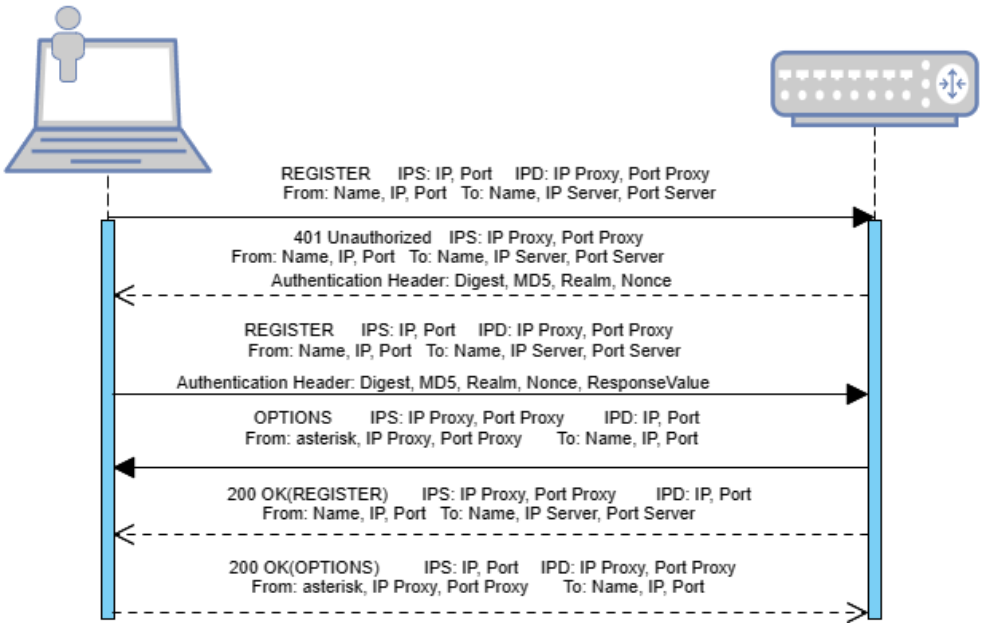

Figura 33. Diagrama de paso de mensajes del registro SIP en el proxy.

Después de la inicialización de los módulos, procederemos a registrar nuestra aplicación en el proxy Asterisk. Para ello, construiremos un mensaje SIP con el método REGISTER en el SCAIP Listener. Antes de eso, debemos configurar las diferentes cabeceras necesarias para la creación del mensaje. Además, guardaremos los valores necesarios en la base de datos para poder reconstruir el mensaje en caso de que sea necesaria una autenticación.

Finalmente, crearemos una Client Transaction para poder enviar el mensaje con estado.

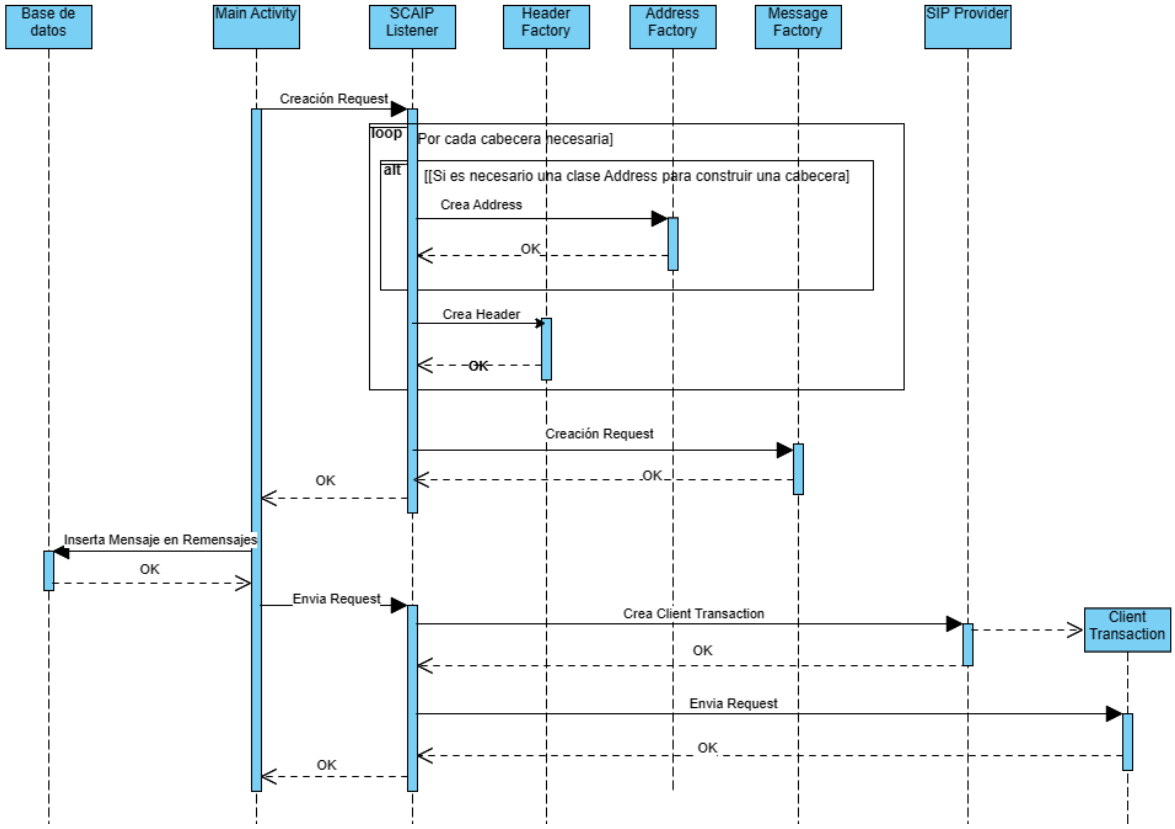

Figura 34. Diagrama de secuencia de la creación y envió de un mensaje

En el proxy, tenemos configurada la autenticación de los mensajes, lo que nos devolverá una respuesta con el

código 401 Unauthorized. Dado que necesitamos responder al desafío de autenticación, calculamos la respuesta correspondiente para crear una cabecera de autenticación.

Una vez calculada la respuesta, recuperamos los valores necesarios de la base de datos para reconstruir el mensaje REGISTER. Para ello, utilizamos el valor único del identificador de llamada (CallID). Después de reconstruir el mensaje, realizamos modificaciones en el CallID y el número de secuencia del mensaje para construir manualmente un mensaje que continúe el diálogo entre la aplicación y el proxy.

Finalmente, creamos una client transacción para poder enviar el mensaje con un estado adecuado al proxy.

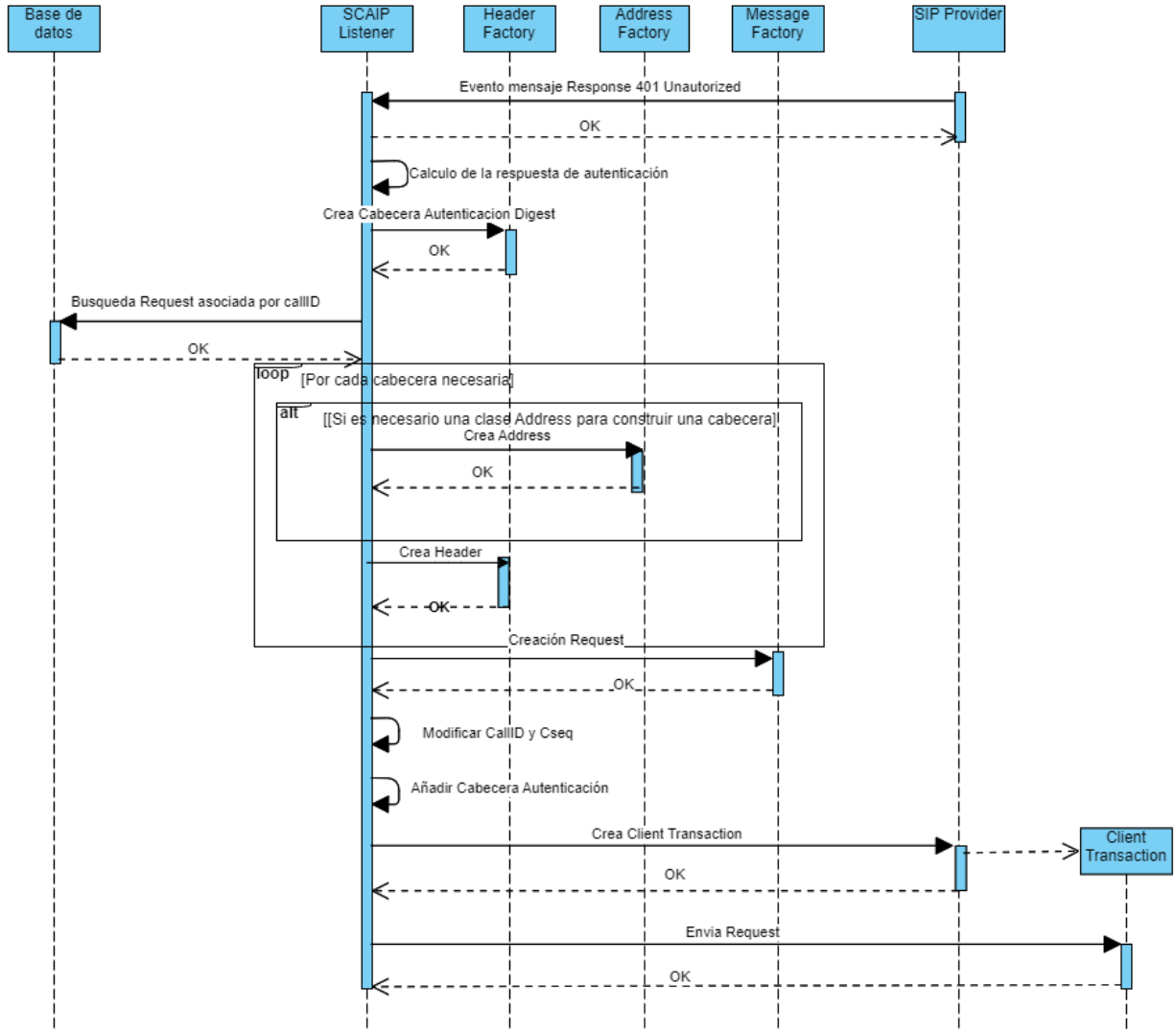

Figura 35. Diagrama de secuencia de autenticación y reenvío de un mensaje

Además de enviar la respuesta exitosa a la solicitud de registro, el proxy nos hace una pregunta sobre las capacidades de nuestra aplicación a través de un mensaje OPTIONS. Este mensaje también se utiliza como una comprobación periódica para verificar si seguimos accesibles.

La aplicación está configurada para responder a cualquier mensaje OPTIONS con un mensaje 200 OK para indicar únicamente nuestra disponibilidad. Para enviar esta respuesta, utilizaremos un objeto Server Transaction que será creado por el SIP Provider.

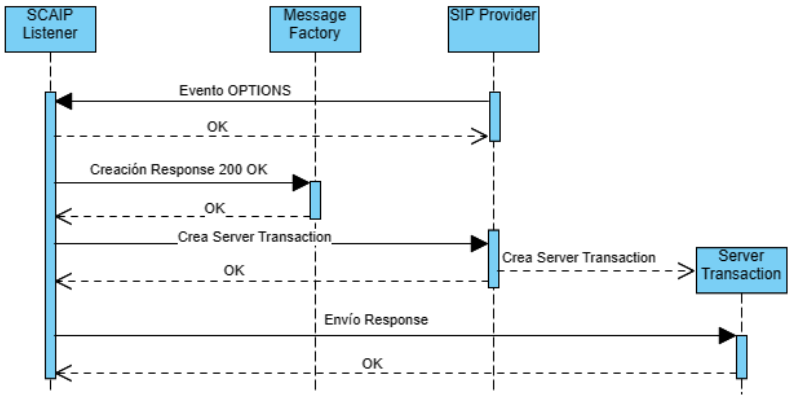

Figura 36. Diagrama de secuencia de la respuesta a un mensaje OPTIONS

```
108 18.706554541 192.168.1.191 192.168.1.251 SIP        498 Request: REGISTER sip:servidorSCAIP@192.168.1.251:5060  (1 binding) |
110 18.706806746 192.168.1.251 192.168.1.191 SIP        676 Status: 401 Unauthorized |
112 18.968626379 192.168.1.191 192.168.1.251 SIP        676 Request: REGISTER sip:servidorSCAIP@192.168.1.251:5060  (1 binding) |
113 18.969669582 192.168.1.251 192.168.1.191 SIP        693 Request: OPTIONS sip:servidorSCAIP@10.0.2.16:5060;transport=tcp |
115 19.010583798 192.168.1.251 192.168.1.191 SIP        712 Status: 200 OK (REGISTER)  (1 binding) |
117 19.109719665 192.168.1.191 192.168.1.251 SIP        443 Status: 200 OK (OPTIONS) |
```

Figura 37. Extracto de Wireshark mostrando el paso de mensajes del registro de SServer

```
137 23.564845315 192.168.1.250 192.168.1.251 SIP        493 Request: REGISTER sip:clienteSCAIP@192.168.1.251:5060  (1 binding) |
139 23.565787091 192.168.1.251 192.168.1.250 SIP        674 Status: 401 Unauthorized |
143 24.002791434 192.168.1.250 192.168.1.251 SIP        669 Request: REGISTER sip:clienteSCAIP@192.168.1.251:5060  (1 binding) |
144 24.004558959 192.168.1.251 192.168.1.250 SIP        691 Request: OPTIONS sip:clienteSCAIP@10.0.2.16:5060;transport=tcp |
146 24.052237603 192.168.1.251 192.168.1.250 SIP        709 Status: 200 OK (REGISTER)  (1 binding) |
148 24.171110753 192.168.1.250 192.168.1.251 SIP        381 Status: 200 OK (OPTIONS) |
```

Figura 38. Extracto de Wireshark mostrando el paso de mensajes del registro de SClock

```
2023-07-08 13:01:31.934 9400-9400/com.example.sclock I/LogMain: Request : REGISTER sip:clienteSCAIP@192.168.1.251:5060 SIP/2.0
    Call-ID: e05c33052549578b14f11d719e3ae77e@10.0.2.16
    CSeq: 1 REGISTER
    From: "clienteSCAIP" <sip:clienteSCAIP@10.0.2.16>;tag=12345
    To: "clienteSCAIP" <sip:clienteSCAIP@192.168.1.251>
    Via: SIP/2.0/TCP 10.0.2.16:5060
    Max-Forwards: 70
    Contact: "clienteSCAIP" <sip:clienteSCAIP@10.0.2.16:5060;transport=tcp>
    Content-Length: 0
2023-07-08 13:01:32.225 9400-9431/com.example.sclock I/LogListener: Response = SIP/2.0 401 Unauthorized
    Via: SIP/2.0/TCP 10.0.2.16:5060;branch=z9hG4bK-313331-ee8ee9f098ebcf914d8e00924201cdf9;received=192.168.1.250;rport=52067
    From: "clienteSCAIP" <sip:clienteSCAIP@10.0.2.16>;tag=12345
    To: "clienteSCAIP" <sip:clienteSCAIP@192.168.1.251>;tag=as4ff87663
    Call-ID: e05c33052549578b14f11d719e3ae77e@10.0.2.16
    CSeq: 1 REGISTER
    Server: Asterisk PBX 18.10.0~dfsg+~cs6.10.40431411-2
    Allow: INVITE,ACK,CANCEL,OPTIONS,BYE,REFER,SUBSCRIBE,NOTIFY,INFO,PUBLISH,MESSAGE
    Supported: replaces,timer
    WWW-Authenticate: Digest algorithm=MD5,realm="asterisk",nonce="32dcdec3"
    Content-Length: 0
2023-07-08 13:01:32.374 9400-9431/com.example.sclock I/LogListener: Request :REGISTER sip:clienteSCAIP@192.168.1.251:5060 SIP/2.0
    From: "clienteSCAIP" <sip:clienteSCAIP@10.0.2.16>;tag=12345
    To: "clienteSCAIP" <sip:clienteSCAIP@192.168.1.251>
    Via: SIP/2.0/TCP 10.0.2.16:5060
    Max-Forwards: 70
    Contact: "clienteSCAIP" <sip:clienteSCAIP@10.0.2.16:5060;transport=tcp>
    Authorization: Digest username="clienteSCAIP",realm="asterisk",nonce="32dcdec3",uri="sip:clienteSCAIP@192.168.1.251",response="f7e3deb7514f5bfea616d2ac102a07c2",algorithm=MD5
    Call-ID: e05c33052549578b14f11d719e3ae77e@10.0.2.16
    CSeq: 2 REGISTER
    Content-Length: 0
```

Figura 39. Trazas del Logcat de Android Studio del registro de SClock (1)

```
2023-07-08 13:01:32.547 9400-9431/com.example.sclock I/LogListener: Request: OPTIONS sip:clienteSCAIP@10.0.2.16:5060;transport=tcp SIP/2.0
    Via: SIP/2.0/TCP 192.168.1.251:5060;branch=z9hG4bK1d499e4a;rport=5060;received=192.168.1.251
    Max-Forwards: 70
    From: "asterisk" <sip:asterisk@192.168.1.251>;tag=as08959172
    To: <sip:clienteSCAIP@10.0.2.16:5060;transport=tcp>
    Contact: <sip:asterisk@192.168.1.251:5060;transport=tcp>
    Call-ID: 50753ad5587fc1aa2520e8ce3234a82e@192.168.1.251:5060
    CSeq: 102 OPTIONS
    User-Agent: Asterisk PBX 18.10.0~dfsg+~cs6.10.40431411-2
    Date: Sat, 08 Jul 2023 11:01:30 GMT
    Allow: INVITE,ACK,CANCEL,OPTIONS,BYE,REFER,SUBSCRIBE,NOTIFY,INFO,PUBLISH,MESSAGE
    Supported: replaces,timer
    Content-Length: 0
2023-07-08 13:01:32.575 9400-9431/com.example.sclock I/LogListener: Response = SIP/2.0 200 OK
    CSeq: 102 OPTIONS
    Call-ID: 50753ad5587fc1aa2520e8ce3234a82e@192.168.1.251:5060
    From: "asterisk" <sip:asterisk@192.168.1.251>;tag=as08959172
    To: <sip:clienteSCAIP@10.0.2.16:5060;transport=tcp>
    Via: SIP/2.0/TCP 192.168.1.251:5060;branch=z9hG4bK1d499e4a;rport=5060;received=192.168.1.251
    Content-Length: 0
2023-07-08 13:01:32.607 9400-9431/com.example.sclock I/LogListener: Response = SIP/2.0 200 OK
    Via: SIP/2.0/TCP 10.0.2.16:5060;branch=z9hG4bK-313331-274f0cc9c1713f7a469a17582677b17b;received=192.168.1.250;rport=52067
    From: "clienteSCAIP" <sip:clienteSCAIP@10.0.2.16>;tag=12345
    To: "clienteSCAIP" <sip:clienteSCAIP@192.168.1.251>;tag=as4ff87663
    Call-ID: e05c33052549578b14f11d719e3ae77e@10.0.2.16
    CSeq: 2 REGISTER
    Server: Asterisk PBX 18.10.0~dfsg+~cs6.10.40431411-2
    Allow: INVITE,ACK,CANCEL,OPTIONS,BYE,REFER,SUBSCRIBE,NOTIFY,INFO,PUBLISH,MESSAGE
    Supported: replaces,timer
    Expires: 120
    Contact: <sip:clienteSCAIP@10.0.2.16:5060;transport=tcp>;expires=120
    Date: Sat, 08 Jul 2023 11:01:30 GMT
    Content-Length: 0
```

Figura 40. Trazas del Logcat de Android Studio del registro de SClock (2)

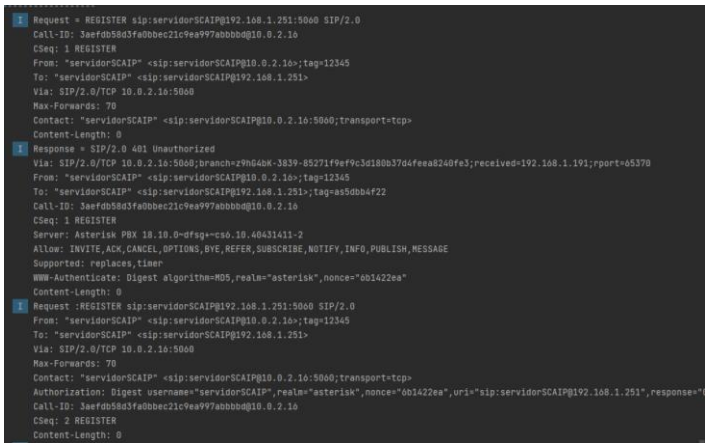

Figura 41. Trazas del Logcat de Android Studio del registro de SServer (1)

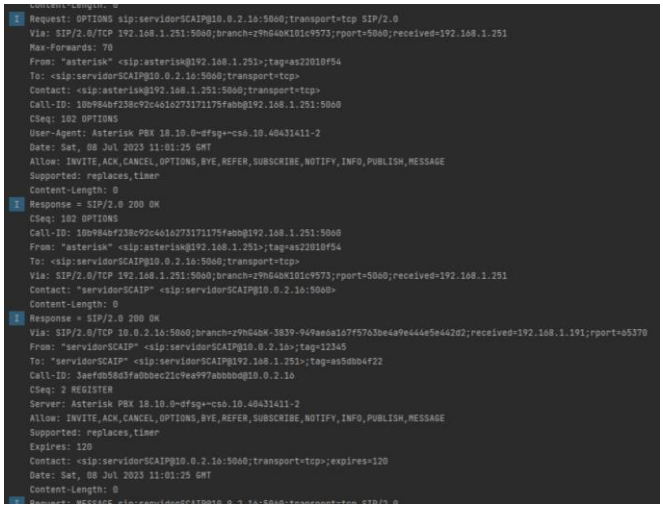

Figura 42. Trazas del Logcat del Android Studio del registro de SServer (2)

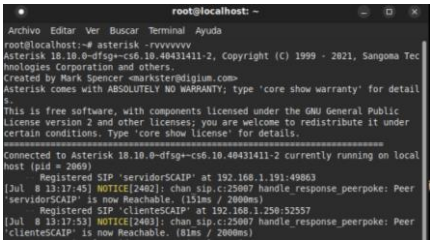

Figura 43. Trazas del Proxy ASTERISK del registro

5.2 Envío de alertas simple

Después de que la aplicación SClock se registre en el proxy pasaremos a la actividad "Lista Botones" la cual creará el menú principal de la aplicación. Este menú estará compuesto por una lista de botones, donde cada botón representa una de las posibles alarmas configuradas. Para lograr esto, se utilizará un RecyclerView.

En primer lugar, la aplicación rellenará la tabla de plantillas en la base de datos. Luego, se comprobará que las plantillas se hayan insertado correctamente recuperándolas de la base de datos. Estas plantillas se utilizarán para crear los botones.

En la actividad principal, se creará un objeto RecyclerView y un adaptador RecyclerView. El adaptador se encargará de cargar los botones en la vista utilizando los datos que se pasan a través de los argumentos, que en este caso son los nombres de las distintas alarmas. Al añadir el adaptador a la vista, se crearán los diferentes botones y se mostrarán en la pantalla.

Cuando el usuario toque uno de los botones, este llamará a una función de la actividad para cambiar al menú asociado a esa alarma. El menú al que se dirija dependerá de si la alarma es manual o automática.

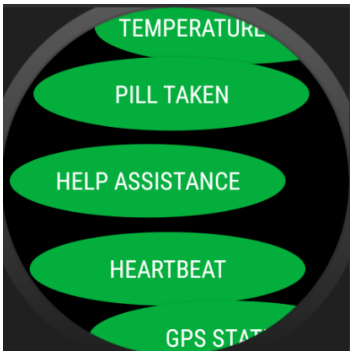

Figura 44. Lista de alarmas de SClock

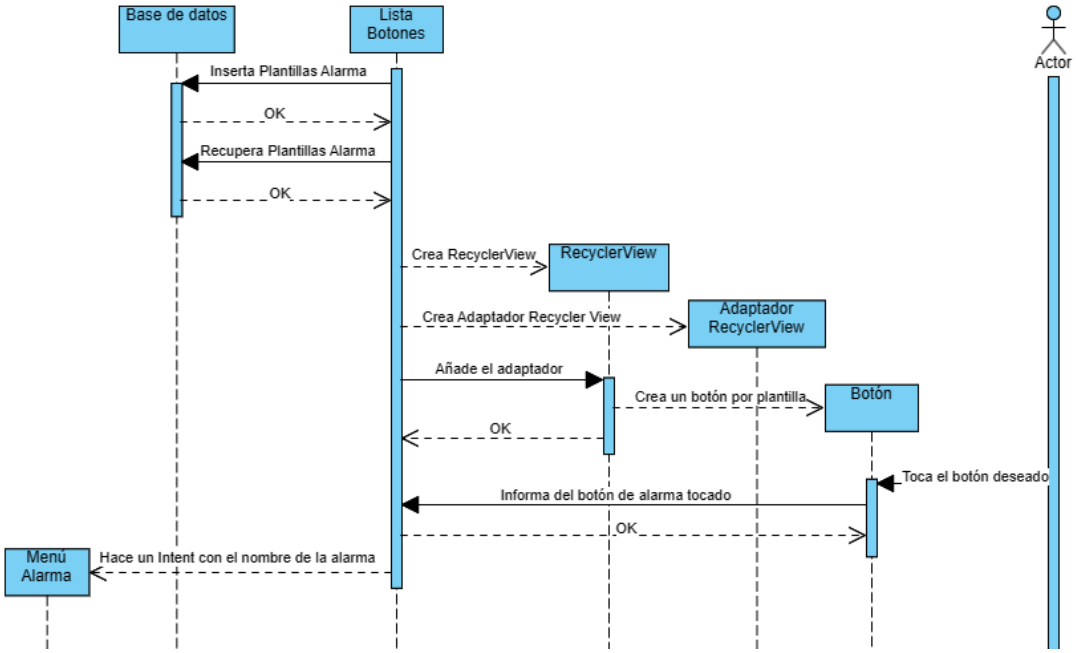

Figura 45. Diagrama de secuencia del menú de las distintas alarmas

En esta sección, asumimos que la alarma seleccionada es de envío manual, lo que nos lleva a la actividad "Simple Alarm View". En esta vista, encontraremos dos botones: uno para regresar atrás y otro para enviar la alarma.

Cuando presionamos el botón de enviar alarma, la actividad llamará a la clase "SCAIP Construct" para crear y validar la solicitud XML. Esta solicitud será enviada dentro de un mensaje SIP tipo "MESSAGE", que se construirá de la misma manera que hemos explicado en la sección anterior. Además, guardaremos la solicitud en la base de datos para su reenvío posterior y crearemos una "Client Transaction" para enviar el mensaje con un estado específico al proxy.

Figura 46. Menú de alarmas simple de SClock

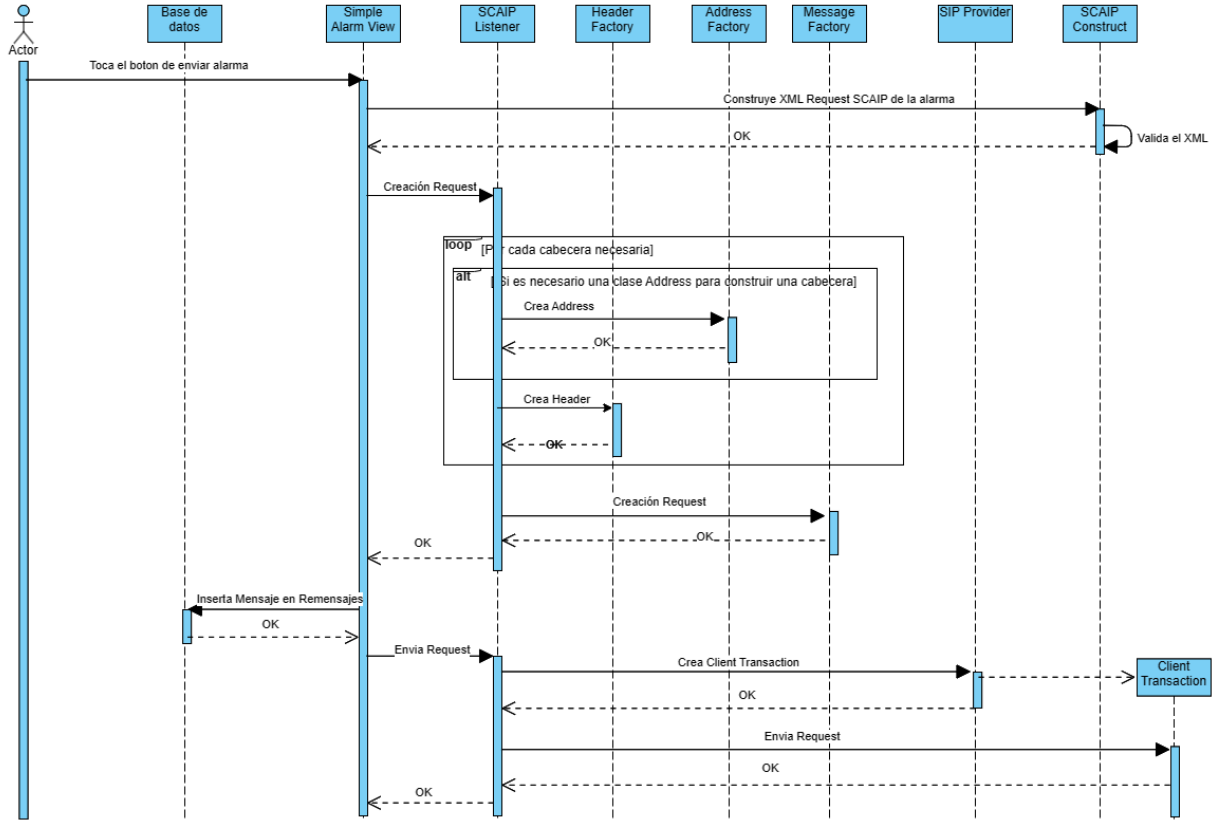

Figura 47. Diagrama de secuencia del envío de mensaje de una alarma manual

El siguiente diagrama muestra todos los mensajes involucrados en la secuencia completa de este caso de uso. Observamos que todos los mensajes que enviamos reciben inicialmente un desafío del servidor para autenticar al emisor del mensaje. Los mensajes que enviamos son solicitudes SCAIP encapsuladas en un mensaje SIP tipo MESSAGE, y las respuestas a estos son respuestas SCAIP encapsuladas en mensajes SIP tipo MESSAGE.

Es importante destacar que, al igual que mencionamos anteriormente, los campos "from" y "to" no cambian entre el mensaje y la respuesta. Además, cabe señalar que no es necesaria la autenticación al recibir mensajes SIP del proxy, solo al enviar mensajes a este.

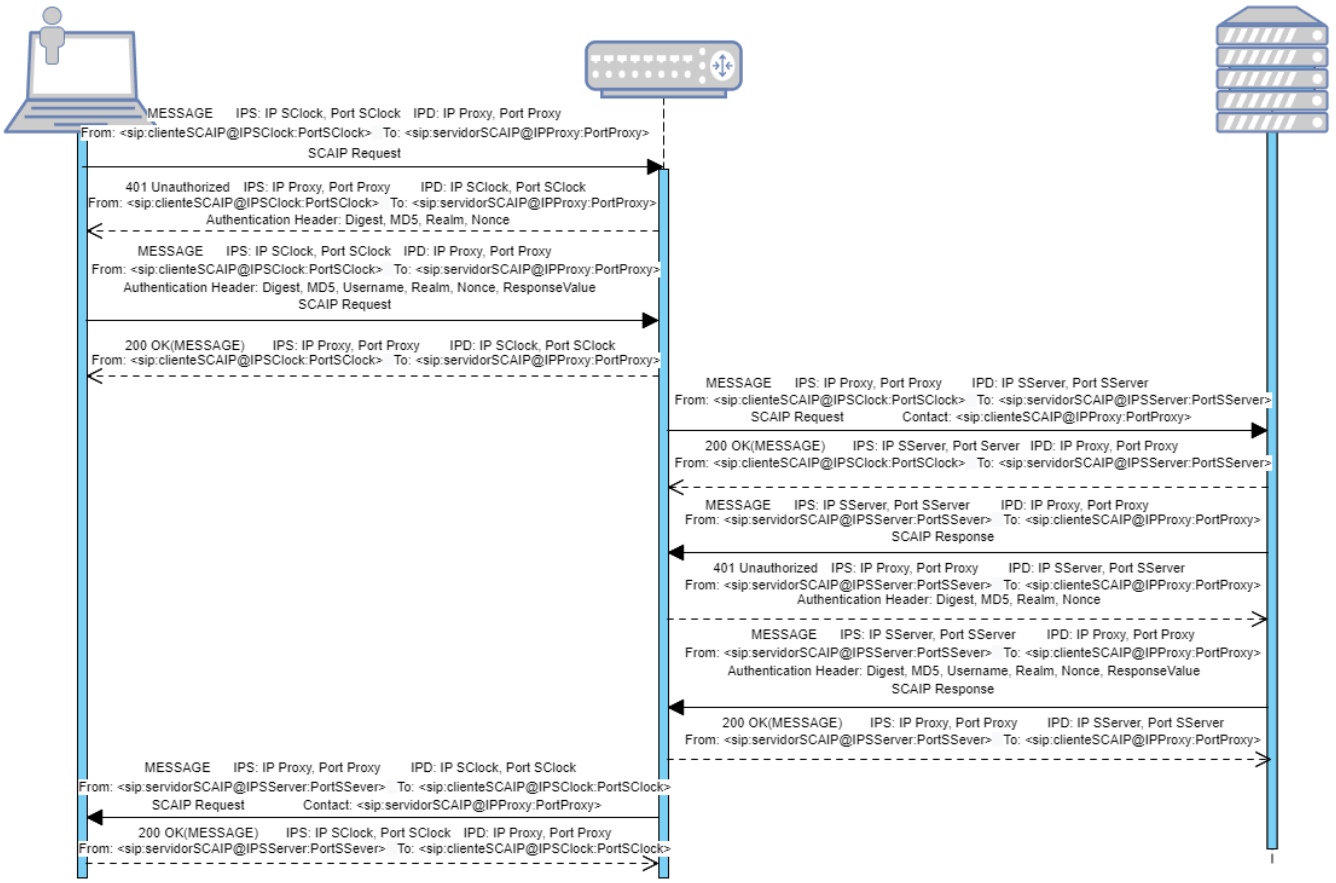

Figura 48. Diagrama de paso de mensaje de una alarma SCAIP

Continuando con la secuencia de este caso de uso, una vez que el mensaje SIP es enviado, el proxy responde con un desafío de autenticación. En este punto, debemos seguir el flujo del diagrama de secuencia de la figura 35 (Diagrama de secuencia de autenticación y reenvío de un mensaje) y reenviar el mensaje incluyendo la respuesta a la autenticación requerida.

El proxy envía el mensaje al servidor correspondiente, el cual responde con una respuesta 200 o a través de un Server Transaction y se procede a analizar el contenido del mensaje. Si el contenido es una solicitud SCAIP, el servidor genera una respuesta SCAIP para informar al cliente que la solicitud ha sido recibida y es correcta. Esta respuesta se encapsula en un mensaje SIP tipo MESSAGE y se guarda en la base de datos para un posible reenvío de la alarma en el futuro.

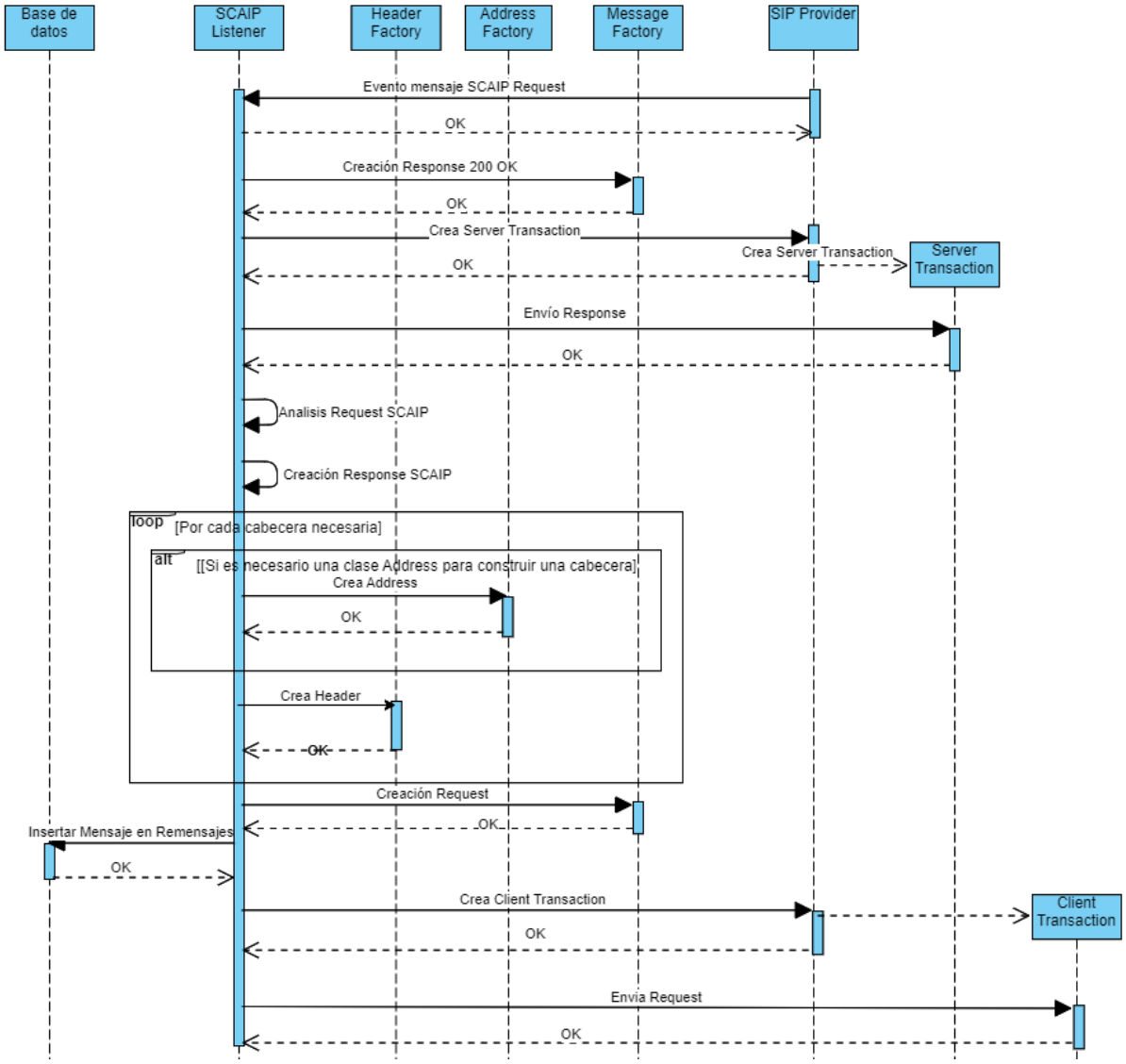

Figura 49. Diagrama de secuencia de respuesta a un mensaje de solicitud (Request) SCAIP

Como esperábamos, el servidor envía un desafío de autenticación en respuesta al mensaje, y el cliente crea la cabecera de autenticación de la misma manera que se muestra en el diagrama de secuencia de la figura 35 (Diagrama de secuencia de autenticación y reenvío de un mensaje).

Finalmente, el proxy reenvía el mensaje al cliente y este confirma la recepción de la misma manera que se muestra en el diagrama de secuencia de la figura 36 (Diagrama de secuencia de la respuesta a un mensaje OPTIONS).

```
248 31.750706585 192.168.1.250 192.168.1.251 SIP      604 Request: MESSAGE sip:servidorSCAIP@192.168.1.251:5060 |  (text/plain)
250 31.751545487 192.168.1.251 192.168.1.250 SIP      675 Status: 401 Unauthorized |
257 31.965329861 192.168.1.250 192.168.1.251 SIP      781 Request: MESSAGE sip:servidorSCAIP@192.168.1.251:5060 |  (text/plain)
258 31.965637011 192.168.1.251 192.168.1.250 SIP      595 Status: 202 Accepted |
259 31.965935718 192.168.1.251 192.168.1.191 SIP      664 Request: MESSAGE sip:servidorSCAIP@10.0.2.16:5060;transport=tcp |  (text/plain)
262 32.049833301 192.168.1.191 192.168.1.251 SIP      443 Status: 200 OK (MESSAGE) |
264 32.172487403 192.168.1.191 192.168.1.251 SIP      560 Request: MESSAGE sip:clienteSCAIP@192.168.1.251:5060 |  (text/plain)
266 32.172794653 192.168.1.251 192.168.1.191 SIP      673 Status: 401 Unauthorized |
268 32.272714733 192.168.1.191 192.168.1.251 SIP      737 Request: MESSAGE sip:clienteSCAIP@192.168.1.251:5060 |  (text/plain)
269 32.273033791 192.168.1.251 192.168.1.191 SIP      593 Status: 202 Accepted |
270 32.273302778 192.168.1.251 192.168.1.250 SIP      622 Request: MESSAGE sip:clienteSCAIP@10.0.2.16:5060;transport=tcp |  (text/plain)
273 32.381538365 192.168.1.250 192.168.1.251 SIP      383 Status: 200 OK (MESSAGE) |

> Frame 257: 781 bytes on wire (6248 bits), 781 bytes captured (6248 bits) on interface enp0s3, id 0
> Ethernet II, Src: IntelCor_57:49:55 (f0:9e:4a:57:49:55), Dst: PcsCompu_dd:02:e8 (08:00:27:dd:02:e8)
> Internet Protocol Version 4, Src: 192.168.1.250, Dst: 192.168.1.251
> Transmission Control Protocol, Src Port: 52067, Dst Port: 5060, Seq: 1932, Ack: 2534, Len: 727
v Session Initiation Protocol (MESSAGE)
  > Request-Line: MESSAGE sip:servidorSCAIP@192.168.1.251:5060 SIP/2.0
  > Message Header
  v Message Body
    v Line-based text data: text/plain (1 lines)
        <mrq><ref>Pil595</ref><cid>836710811199107</cid><dty>0002</dty><stc>001</stc></mrq>
```

Figura 50. Extracto de Wireshark del paso de mensajes del envío de una alarma manual

```
2023-07-08 13:01:40.188 9400-9400/com.example.sclock I/SimpleAlarmView: Request : MESSAGE sip:servidorSCAIP@192.168.1.251:5060 SIP/2.0
    Call-ID: 602adaa9d04da89cd17a9fe7e38d2e93@10.0.2.16
    CSeq: 1 MESSAGE
    From: "clienteSCAIP" <sip:clienteSCAIP@10.0.2.16>;tag=12345
    To: "servidorSCAIP" <sip:servidorSCAIP@192.168.1.251>
    Via: SIP/2.0/TCP 10.0.2.16:5060
    Max-Forwards: 70
    Contact: "clienteSCAIP" <sip:clienteSCAIP@10.0.2.16:5060;transport=tcp>
    Content-Type: text/plain
    Content-Length: 83

    <mrq><ref>Pil595</ref><cid>836710811199107</cid><dty>0002</dty><stc>001</stc></mrq>
2023-07-08 13:01:40.252 9400-9431/com.example.sclock I/LogListener: Response = SIP/2.0 401 Unauthorized
    Via: SIP/2.0/TCP 10.0.2.16:5060;branch=z9hG4bK-313331-6e02d813dbb71912f974cbc9857d8022;received=192.168.1.250;rport=52067
    From: "clienteSCAIP" <sip:clienteSCAIP@10.0.2.16>;tag=12345
    To: "servidorSCAIP" <sip:servidorSCAIP@192.168.1.251>;tag=as754e4b55
    Call-ID: 602adaa9d04da89cd17a9fe7e38d2e93@10.0.2.16
    CSeq: 1 MESSAGE
    Server: Asterisk PBX 18.10.0-dfsg+~cs6.10.40431411-2
    Allow: INVITE,ACK,CANCEL,OPTIONS,BYE,REFER,SUBSCRIBE,NOTIFY,INFO,PUBLISH,MESSAGE
    Supported: replaces,timer
    WWW-Authenticate: Digest algorithm=MD5,realm="asterisk",nonce="1e1b8179"
    Content-Length: 0
2023-07-08 13:01:40.351 9400-9431/com.example.sclock I/LogListener: Request :MESSAGE sip:servidorSCAIP@192.168.1.251:5060 SIP/2.0
    From: "clienteSCAIP" <sip:clienteSCAIP@10.0.2.16>;tag=12345
    To: "servidorSCAIP" <sip:servidorSCAIP@192.168.1.251>
    Via: SIP/2.0/TCP 10.0.2.16:5060
    Max-Forwards: 70
    Contact: "clienteSCAIP" <sip:clienteSCAIP@10.0.2.16:5060;transport=tcp>
    Content-Type: text/plain
    Authorization: Digest username="clienteSCAIP",realm="asterisk",nonce="1e1b8179",uri="sip:servidorSCAIP@192.168.1.251",response="ff53dbb08b7f7de53711ce0a3ac73dc4",algorithm=MD5
    Call-ID: 602adaa9d04da89cd17a9fe7e38d2e93@10.0.2.16
    CSeq: 2 MESSAGE
    Content-Length: 83

    <mrq><ref>Pil595</ref><cid>836710811199107</cid><dty>0002</dty><stc>001</stc></mrq>
```

Figura 51. Trazas del Logcat de Android Studio del envío de mensajes manuales de SClock (1)

```
2023-07-08 13:01:40.589 9400-9431/com.example.sclock I/LogListener: Response = SIP/2.0 202 Accepted
    Via: SIP/2.0/TCP 10.0.2.16:5060;branch=z9hG4bK-313331-d58ef71b945b705e5cdcfc323ecb90e2;received=192.168.1.250;rport=52067
    From: "clienteSCAIP" <sip:clienteSCAIP@10.0.2.16>;tag=12345
    To: "servidorSCAIP" <sip:servidorSCAIP@192.168.1.251>;tag=as754e4b55
    Call-ID: 602adaa9d04da89cd17a9fe7e38d2e93@10.0.2.16
    CSeq: 2 MESSAGE
    Server: Asterisk PBX 18.10.0-dfsg+~cs6.10.40431411-2
    Allow: INVITE,ACK,CANCEL,OPTIONS,BYE,REFER,SUBSCRIBE,NOTIFY,INFO,PUBLISH,MESSAGE
    Supported: replaces,timer
    Content-Length: 0
2023-07-08 13:01:40.719 9400-9431/com.example.sclock I/LogListener: Request: MESSAGE sip:clienteSCAIP@10.0.2.16:5060;transport=tcp SIP/2.0
    Via: SIP/2.0/TCP 192.168.1.251:5060;branch=z9hG4bK2c9047e8;rport=5060;received=192.168.1.251
    Max-Forwards: 70
    From: "servidorSCAIP" <sip:servidorSCAIP@10.0.2.16>;tag=as3a11398d
    To: <sip:clienteSCAIP@10.0.2.16:5060;transport=tcp>
    Contact: <sip:servidorSCAIP@192.168.1.251:5060;transport=tcp>
    Call-ID: 5023509b0024fa592a718c5d75e44371@127.0.0.1:5060
    CSeq: 102 MESSAGE
    User-Agent: Asterisk PBX 18.10.0-dfsg+~cs6.10.40431411-2
    Content-Type: text/plain;charset=UTF-8
    Content-Length: 40

    <mrs><ref>Pil595</ref><snu>0</snu></mrs>
2023-07-08 13:01:40.785 9400-9431/com.example.sclock I/LogListener: clienteSCAIP: Response = SIP/2.0 200 OK
    CSeq: 102 MESSAGE
    Call-ID: 5023509b0024fa592a718c5d75e44371@127.0.0.1:5060
    From: "servidorSCAIP" <sip:servidorSCAIP@10.0.2.16>;tag=as3a11398d
    To: <sip:clienteSCAIP@10.0.2.16:5060;transport=tcp>
    Via: SIP/2.0/TCP 192.168.1.251:5060;branch=z9hG4bK2c9047e8;rport=5060;received=192.168.1.251
    Content-Length: 0
```

Figura 52. Trazas del Logcat de Android Studio del envío de mensajes manuales de SClock (2)

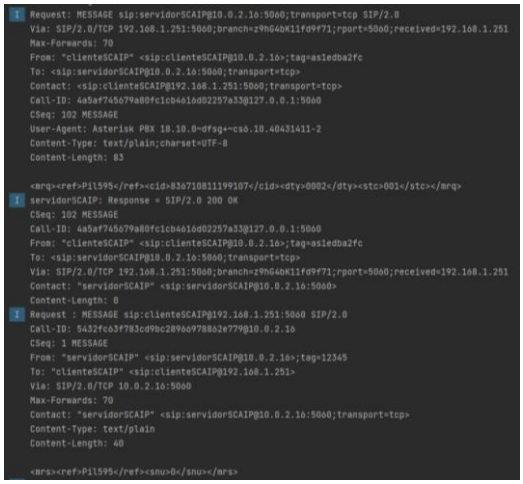

Figura 53. Trazas del Logcat de Android Studio del envío de mensajes manuales de SServer (1)

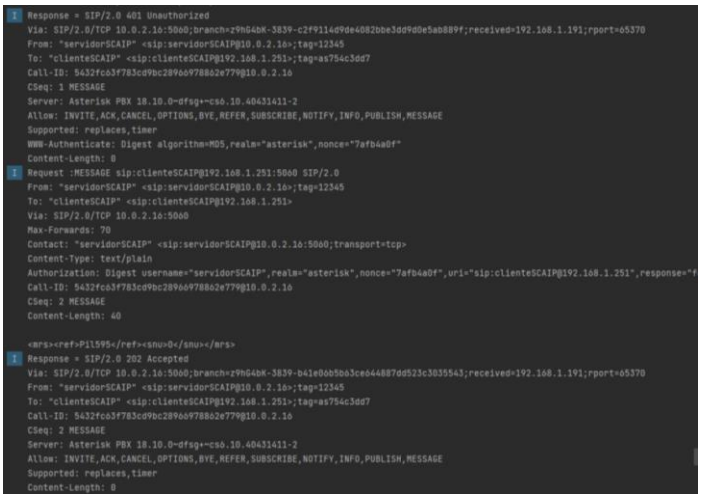

Figura 54. Trazas del Logcat de Android Studio del envío de mensajes manuales de SServer (2)

5.3 Programación y envío de alarmas programadas

Si en la lista de botones de la pantalla principal hubiéramos seleccionado una alarma de envío automático, nos habría redirigido a la actividad "Choice Alarm View". Esta actividad muestra 5 botones en pantalla para configurar el periodo de envío de la alarma seleccionada.

Cuando el actor pulsa uno de los botones, si se trata de un botón con un periodo de envío positivo, la actividad recuperará la plantilla correspondiente a esa alarma de la base de datos y configurará una alarma automática en la tabla de alarmas automáticas. Si ya existiera una alarma programada previamente, esta se sobrescribirá o se informará del error en caso de que ocurra algún problema.

En cambio, si el actor pulsa el botón "0", la actividad accederá a la base de datos para eliminar la tupla de la alarma programada anteriormente. En caso de que esa alarma no estuviera programada, se mostrará un mensaje de error para informar al actor sobre esta situación.

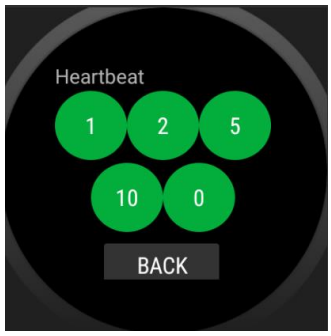

Figura 55. Menú de configuración de alarmas automáticas

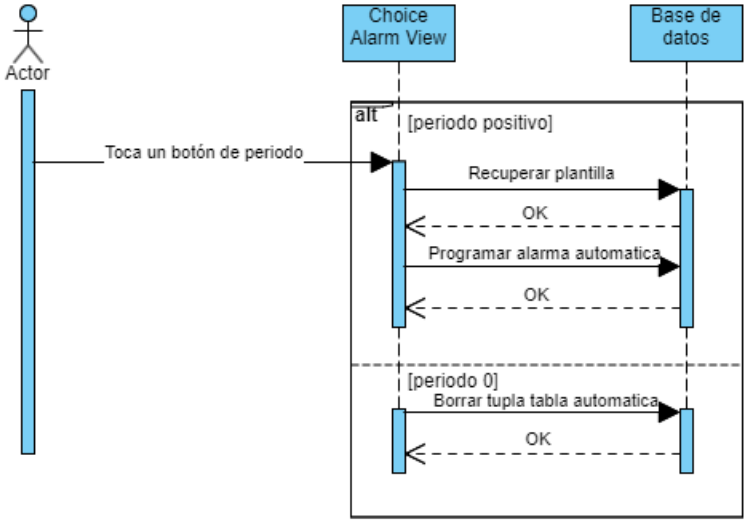

Figura 56. Diagrama de secuencia de configuración de alarmas automáticas

El funcionamiento del envío de alarmas programadas se basa en la inicialización del módulo SCAIP AutoMessager. Este módulo, después de un retraso inicial, genera un hilo (thread) que se ejecuta de forma continua. En cada ciclo, el módulo accede a la base de datos para recuperar las tuplas de la tabla de alarmas automáticas y verifica si alguna de estas alarmas cumple el periodo de envío programado. Si alguna alarma cumple las condiciones, se activa la lógica de envío correspondiente a esa alarma.

El módulo SCAIP Construct se encarga de crear la solicitud SCAIP necesaria para el envío de la alarma programada. Esta solicitud se encapsula en un mensaje SIP tipo MESSAGE y se envía al destino correspondiente. Para obtener más detalles sobre la lógica de creación y envío, se puede consultar la figura 47 (Diagrama de secuencia del envío de mensaje de una alarma manual).

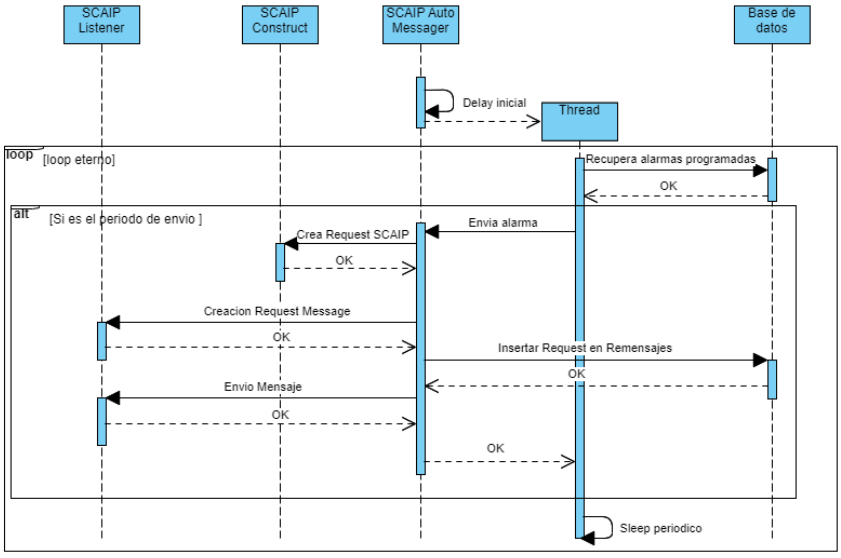

Figura 57. Diagrama de secuencia del envío de una alarma programada

El flujo de mensajes y la secuencia de este proceso a partir de este momento es idéntico al de las alarmas manuales.

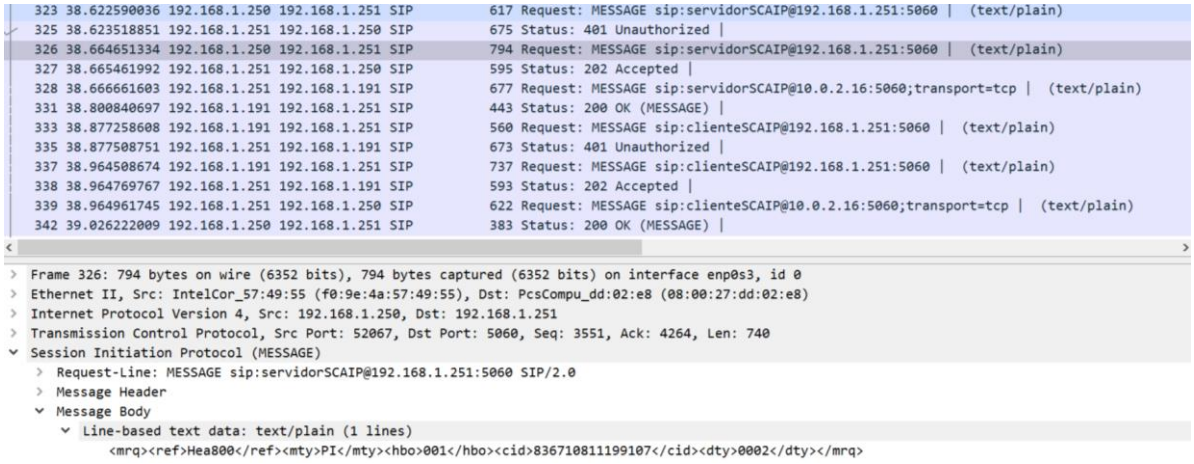

Figura 58. Extracto de Wireshark del paso de mensajes del envío de una alarma automática

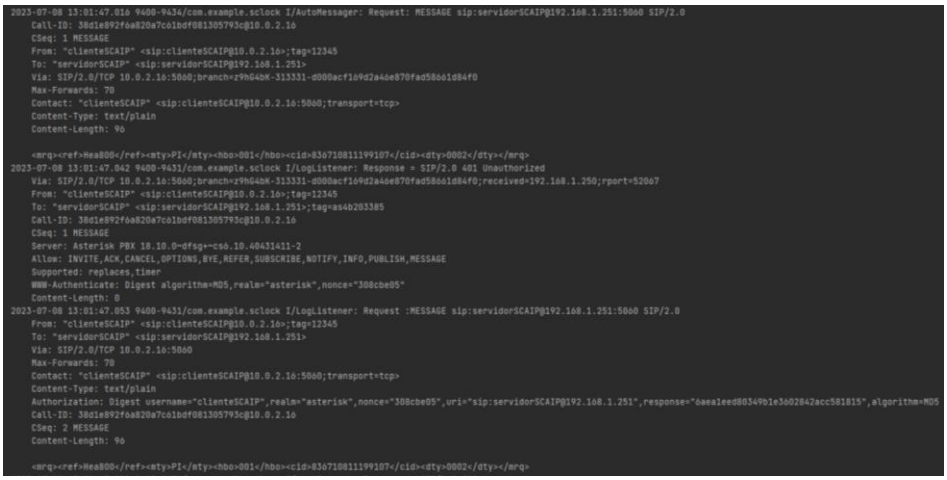

Figura 59. Trazas del Logcat de Android Studio del envío de mensajes automáticos de SClock (1)

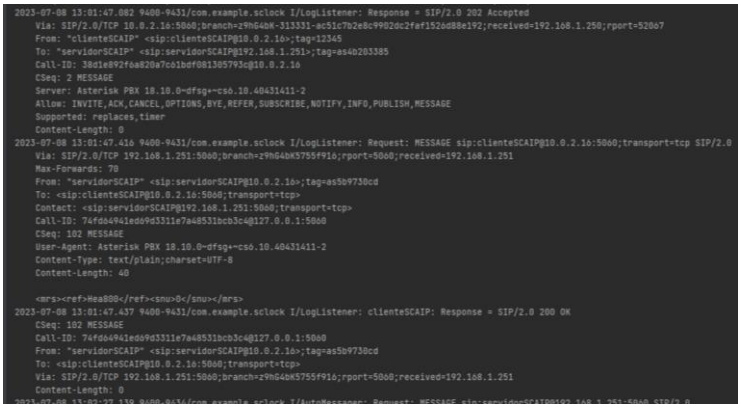

Figura 60. Trazas del Logcat de Android Studio del envío de mensajes automáticos de SClock (2)

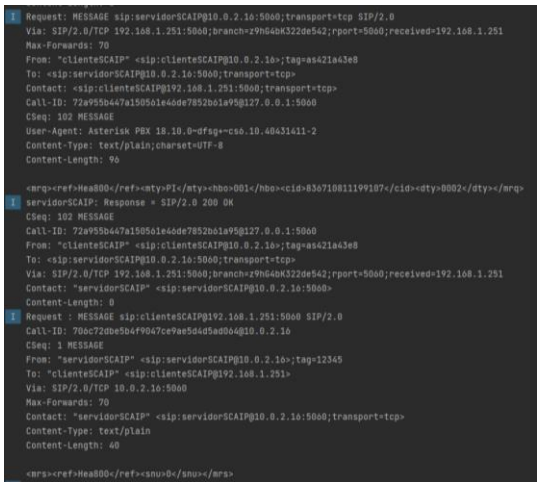

Figura 61. Trazas del Logcat de Android Studio del envío de mensajes automáticos de SServer (1)

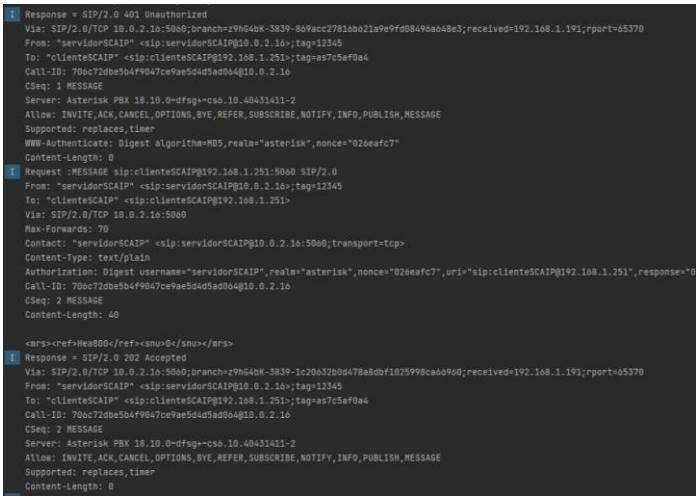

Figura 62. Trazas del Logcat de Android Studio del envío de mensajes automáticos de SServer (2)

5.4 TLS

En esta parte de las pruebas vamos a comprobar que se puede usar el protocolo TLS para las comunicaciones a través del registro de una aplicación a través del protocolo. Vemos como en las capturas Wireshark muestra cómo, a través del protocolo TLS, hace el intercambio de claves públicas y se envían de manera cifrada los mensajes entre el servidor y el proxy mientras que en las trazas de Android Studio podemos ver los mensajes descifrados.

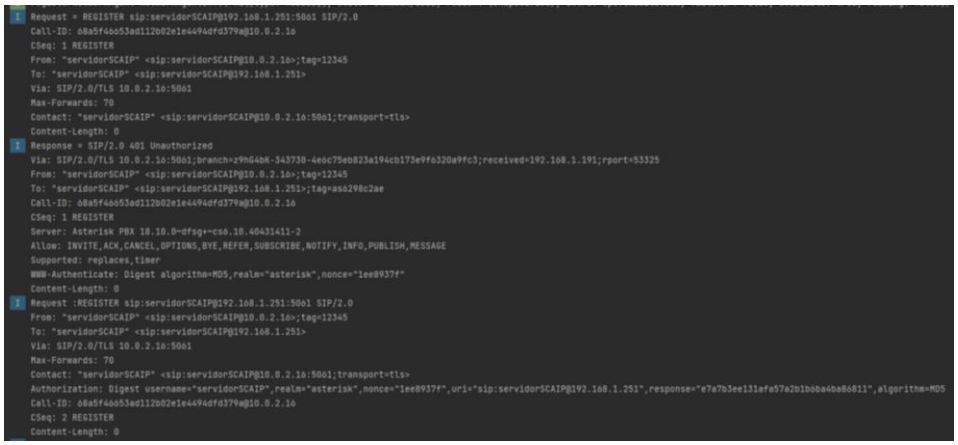

Figura 63. Trazas del Logcat de Android Studio del registro de SServer a través de TLS (1)

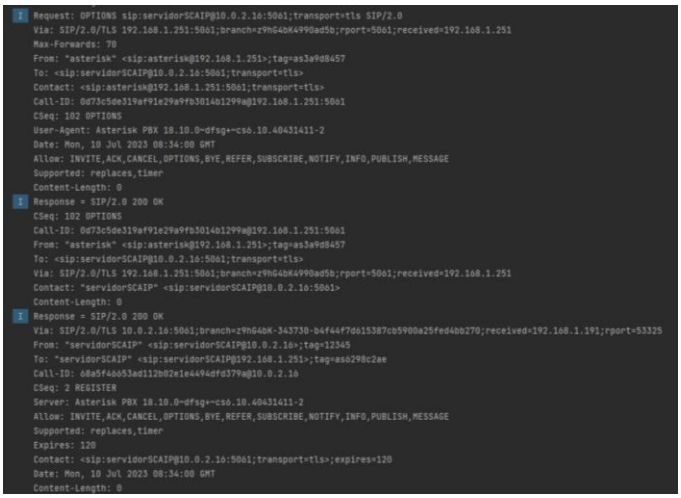

Figura 64. Trazas del Logcat de Android Studio del registro de SServer a través de TLS (2)

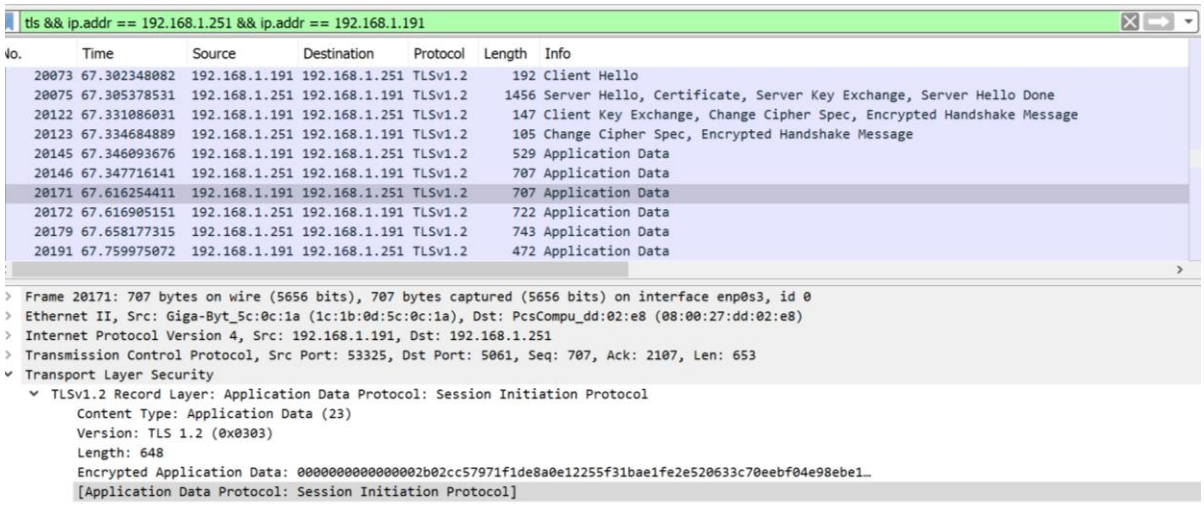

Figura 65. Extracto de Wireshark del paso de mensajes del registro de SServer a través de TLS

6 CONCLUSIONES

6.1 Mejoras

Debido a la magnitud del proyecto, no hemos profundizado en ciertos aspectos del trabajo para evitar perder el rumbo. Uno de los aspectos que podríamos haber explorado más a fondo es el segundo caso de uso de SCAIP, que involucra comunicación de voz. Esto nos habría permitido investigar la comunicación por voz en detalle, utilizando métodos SIP diferentes a los que ya hemos usado, como INVITE, BYE y CANCEL. Además, este caso de uso nos habría dado la oportunidad de explorar más la librería JAIN SIP, que incluye objetos que facilitan los diálogos. Estos objetos podrían haberse utilizado al final del proyecto, al implementar la autenticación digest, sin embargo, logramos conseguir esta funcionalidad manualmente.

En la aplicación de pruebas, SCAIPPA, también habíamos explorado la posibilidad de registrar todos los mensajes SIP recibidos en una base de datos para analizar los diferentes diálogos de la aplicación. Originalmente, planeábamos agregar esta funcionalidad a la aplicación de reloj, accediendo a ella a través de la actividad principal. Sin embargo, debido al tamaño reducido de la pantalla del reloj, resultaba difícil de leer.

Además, consideramos otras mejoras, como leer los datos de configuración desde archivos de texto guardados en la carpeta de recursos, en lugar de utilizar variables codificadas en el propio código. Por ejemplo, esto se aplicaría a los datos de inicialización del SIP Listener y a la carga de las plantillas de los mensajes SCAIP.

También habíamos planeado incorporar una funcionalidad menor que involucrara aprovechar la información de los sensores GPS de los dispositivos Android para enviar alarmas con datos de ubicación GPS.

6.2 Problemas

Durante el desarrollo de este proyecto, nos encontramos con varios problemas. El primero de ellos surgió al implementar la librería JAIN SIP, el cual daba problemas incoherentes. Esto se debía a que la versión de la librería era para Java y, para poder utilizarla en Android, era necesario utilizar una versión modificada específicamente para Android. Esta información no estaba en la documentación y lo aprendí recurriendo a ejemplos y recursos de terceros. Además, al utilizar esta versión modificada, fue necesario ajustar algunos nombres de archivos debido a la necesidad de agregar el prefijo "android" el cual se volvió tedioso.

Otro problema que encontré al implementar la librería fue relacionado con la herramienta de depuración integrada log4j. Aunque no utilicé esta funcionalidad, su implementación causó dificultades, ya que requería la inclusión de bibliotecas de dependencia adicionales. No logré hacer que esta herramienta funcionara correctamente, sospecho que porque no estaba bien adaptada para Android. Esto significó que, en lugar de tener la comodidad de realizar la depuración mediante comentarios en la consola, tuve que depurar paso a paso utilizando la herramienta de depuración de Android para verificar qué estaba fallando.

6.3 Lecciones Aprendidas

En términos de conocimiento, he podido profundizar en los conocimientos adquiridos en diversas asignaturas de mi carrera, especialmente en ingeniería de software y desarrollo de aplicaciones móviles. Gracias a estos conocimientos, pude desarrollar este proyecto siguiendo buenas prácticas de desarrollo, y he comprendido la importancia de la investigación y planificación en el desarrollo de cualquier trabajo. También he apreciado la cantidad de tiempo que se puede ahorrar al realizar una investigación adecuada y como las asignaturas de desarrollo de software me han brindado prácticas de programación que me han facilitado el desarrollo.

En cuanto a la vida personal, este trabajo me ha demostrado las habilidades que he adquirido a lo largo de los años, como la resiliencia y otros atributos necesarios para llevar a cabo con éxito un proyecto. También me ha dado una visión del futuro, mostrándome cómo el trabajo puede presentarme situaciones similares en las que seguiré utilizando y mejorando estas habilidades. Además, me ha enseñado que, por mucho que me apasione

un tema, siempre habrá altibajos. Habrá momentos en los que tendré que esforzarme al máximo para seguir trabajando o resolver problemas aparentemente insolubles, pero también habrá momentos de felicidad que solo se obtienen cuando se logra un resultado positivo gracias a los esfuerzos realizados.

Además, una de las lecciones más importantes que he aprendido es ser realista. Aunque se quiera alcanzar la perfección y ofrecer todas las funcionalidades planeadas hay que entender que es imposible. Es necesario saber cuándo detenerse, cuál es el nivel de calidad suficiente y cuándo es el momento de pasar a la siguiente tarea.

6.4 Conclusión

Para mí, el objetivo real de este trabajo era enfrentarme solo a un desafío de gran magnitud y poder aplicar los conocimientos y habilidades que he adquirido durante mi carrera. Al finalizar, siento que he logrado ese objetivo y estoy satisfecho con el trabajo que he realizado, aunque con esa espina de que se podría hacer más.

Quiero concluir agradeciendo una vez más a todas las personas que me han acompañado en este camino. Sin ellos, no tendría motivación para hacer todo lo que hago. Y si estás leyendo esto, también quiero agradecerte por el esfuerzo que has dedicado al leer hasta aquí.

¡Muchas gracias a todos!

REFERENCIAS

[1] «RFC 2543», 14 de junio de 2022. https://www.ietf.org/rfc/rfc2543.txt (accedido 14 de junio de 2022).

[2] H. Schulzrinne, J. Rosenberg, B. Campbell, D. M. Gurle, y C. Huitema, «Session Initiation Protocol (SIP) Extension for Instant Messaging», Internet Engineering Task Force, Request for Comments RFC 3428, dic. 2002. doi: 10.17487/RFC3428.

[3] Swedish Standars Institut, «Digital social alarm - Social care alarm internet protocol (SCAIP) - Specification».

[4] Swedish Standars Institut, «Digital social alarm - Social care alarm internet protocol (SCAIP) - Implementation guideline».

[5] «PJSIP Project Online Documentation — PJSIP Project 2.13-dev documentation». https://docs.pjsip.org/en/latest/ (accedido 22 de agosto de 2022).

[6] «Liblinphone | Linphone». https://www.linphone.org/technical-corner/liblinphone (accedido 22 de agosto de 2022).

[7] «The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 32». https://jcp.org/en/jsr/detail?id=32 (accedido 22 de agosto de 2022).

[8] «O'Doherty y Ranganathan - 2003 - Serving the Developer Community.pdf». Accedido: 22 de septiembre de 2022. [En línea]. Disponible en: https://www.oracle.com/technetwork/java/jain-sip-tutorial-149998.pdf

[9] «jain-sip-ri 1.3.0-91 javadoc (javax.sip)». https://javadoc.io/doc/javax.sip/jain-sip-ri/latest/index.html (accedido 8 de octubre de 2022).

[10] «Kamailio Documentation | The Kamailio SIP Server Project», 6 de marzo de 2010. https://www.kamailio.org/w/documentation/ (accedido 22 de agosto de 2022).

[11] «openSIPS | About / About». https://www.opensips.org/About/About (accedido 22 de agosto de 2022).

[12] «Home - Asterisk Documentation». https://docs.asterisk.org/20/ (accedido 9 de agosto de 2022).

[13] «XML Schema Part 0: Primer Second Edition». https://www.w3.org/TR/xmlschema-0/ (accedido 11 de febrero de 2023).

7 ANEXO

7.1 Guía de instalación

En caso de que la aplicación no haya sido proporcionada, puede obtenerse desde el siguiente enlace en GitHub: https://github.com/GermanBlas/SCAIP-TFG.

Android Studio

Android Studio es la herramienta que hemos utilizado para programar y ejecutar nuestra aplicación. Puedes descargar este programa desde la página principal en https://developer.android.com/studio. La instalación predeterminada es suficiente y puede utilizarse sin realizar configuraciones adicionales.

Una vez instalado, simplemente debemos acceder a la carpeta de la aplicación y abrir la aplicación. Sin embargo, para ejecutarla, necesitaremos crear un emulador en Android Studio. Para hacer esto, debemos acceder al menú "Device Manager" y seleccionar la opción de crear un dispositivo. En las pruebas se ha utilizado un emulador Wear OS Large Round con API Android 30. Es posible que si es la primera vez que utilizas Android Studio, tengas que esperar a que se descarguen algunos componentes adicionales.

Después de completar los pasos anteriores, simplemente debes ejecutar la aplicación a través de la barra de tareas o en Run > Run. Se recomienda observar las trazas en View > Tool Windows > Logcat para obtener información sobre los mensajes enviados.

Proxy ASTERISK

Si se desea comprobar el escenario entero tenemos que descargar Asterisk, el cual vamos a ejecutar en Linux. Para ello hemos utilizado una máquina virtual proporcionada por la escuela. El siguiente enlace contiene las instrucciones para obtenerse: http://trajano.us.es/~fjfj/mv/maq_virtual.html.

Hemos utilizado Oracle VM VirtualBox para montar la máquina virtual. Para que la máquina tenga acceso directo a la red es necesario entrar en los ajustes de red la MV en Oracle y configurar el adaptador de red para que esté conectado a "Adaptador puente". Con esta opción, si la aplicación SClock o SServer se encuentran alojada en el mismo ordenador, es posible que los paquetes de red no se vean correctamente. Por lo tanto, se recomienda revisar los paquetes en Wireshark dentro de la MV de Asterisk, a través de su interfaz virtual.

Una vez se haya configurado la máquina virtual, procederemos a instalar Asterisk utilizando los repositorios con el comando "apt-get install asterisk". Después de la instalación y al tenerlo en funcionamiento, realizaremos modificaciones en los archivos principales de configuración: sip.conf y extension.conf, que se encuentran en la carpeta /etc/asterisk/. Para obtener instrucciones detalladas sobre estos cambios consultar la sección 3.3.3 de la memoria.

Además, si se desea utilizar la funcionalidad TLS, tendremos que crear los certificados necesarios para ello. Los pasos detallados para construir el certificado para asterisk se encuentran en la sección 3.3.3.3 del TFG. Para las aplicaciones, los certificados de SClock y SServer ya se encuentran incorporados en la propia aplicación. Aun así, si se quieren crear unos nuevos, el procedimiento se detalla en el apartado 4.1.5.3.

Tras realizar cambios en las configuraciones, es necesario reiniciar el programa utilizando el comando "service asterisk restart". Para acceder a la consola de Asterisk, puede utilizarse el comando "asterisk -rv". Cuantas más "v" se añadan, más información detallada mostrará la aplicación en la pantalla. Este comando puede ser de interés para, con las suficientes "v", ver las trazas que genera el programa para ver la interacción con los mensajes SIP generados. Siempre puedes verificar el correcto funcionamiento de la aplicación descargando un software de telefonía SIP. En nuestro caso, hemos utilizado Zoiper, que se puede instalar en dispositivos móviles.

7.2 Código de Android

7.2.1 Main Activity

```
package com.example.sclock.Vistas;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.gov.nist.javax.sip.header.CSeq;
import android.gov.nist.javax.sip.header.CallID;
import android.javax.sip.message.Request;
import android.net.wifi.WifiManager;
import android.os.Bundle;
import android.text.format.Formatter;
import android.util.Log;
import android.view.View;


import com.example.sclock.R;
import com.example.sclock.SCAIP.SCAIPAutoMessager;
import com.example.sclock.SCAIP.SCAIPConstruct;
import com.example.sclock.SCAIP.SCAIPListener;
import com.example.sclock.SCAIP.SCAIPWebApplication;
import com.example.sclock.basedatos.MiBaseDatos;
import com.example.sclock.basedatos.Remensaje;

import org.xml.sax.SAXException;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import javax.xml.XMLConstants;
import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;

public class MainActivity extends Activity {


    private SCAIPListener SCAIPListener;
    private SCAIPWebApplication global;
    private SCAIPConstruct SCAIPConstruct;
    private MiBaseDatos MDB;

    private SCAIPAutoMessager SCAIPAutoMessager;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        global = ((SCAIPWebApplication) getApplicationContext());
        MDB = new MiBaseDatos(getApplicationContext());
        MDB.borrarTodo();

        //Inicializar SCAIPConstruct
        try {
            Source schemaFile = new
StreamSource(getResources().openRawResource(R.raw.xsd_scaip));
            SchemaFactory factory =
SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
            Schema schema = factory.newSchema(schemaFile);
            SCAIPConstruct = new SCAIPConstruct("01.00","836710811199107", "0002", schema);
            SCAIPConstruct.MDBinsert(MDB);
            global.setSCAIPConstruct(SCAIPConstruct);
        } catch (SAXException e) {
            Log.e("MainActivity", "Error :" + e);
        }

        //Guardamos los certificados en el almacenamiento del movil
```

```
//Primero la keyStore
try {
    InputStream keyStoreInputStream = getResources().openRawResource(R.raw.keystore);
    String directoryPath = getFilesDir().getPath();
    String keyStoreName = "keystore.bks";
    String keystorePath = directoryPath + "/" + keyStoreName;

    // Crea el archivo en el sistema de archivos
    File keyStore = new File(keystorePath);
    try {
        OutputStream keyStoreOutputStream = new FileOutputStream(keyStore);
        byte[] buffer = new byte[1024];
        int length;
        while ((length = keyStoreInputStream.read(buffer)) > 0) {
            keyStoreOutputStream.write(buffer, 0, length);
        }
        keyStoreOutputStream.close();
        keyStoreInputStream.close();
    } catch (IOException e) {
        Log.e("LogMain", "Error :" + e);
    }

    //Despues la trustStore
    InputStream trustStoreInputStream = getResources().openRawResource(R.raw.truststore);
    String trustStoreName = "truststore.bks";
    String trustStorePath = directoryPath + "/" + trustStoreName;

    // Crea el archivo en el sistema de archivos
    File trustStore = new File(trustStorePath);

    try {
        OutputStream trustStoreOutputStream = new FileOutputStream(trustStore);
        byte[] buffer = new byte[1024];
        int length;
        while ((length = trustStoreInputStream.read(buffer)) > 0) {
            trustStoreOutputStream.write(buffer, 0, length);
        }
        trustStoreOutputStream.close();
        trustStoreInputStream.close();
    } catch (IOException e) {
        Log.e("LogMain", "Error: " + e);
    }

    String keyStoreType = "BKS";
    String keyStorePassword = "SCAIPPa";
    String trustStoreType = "BKS";
    String trustStorePassword = "SCAIPPa";

    //Inicializamos las propiedades de la conexión SSL
    System.setProperty( "javax.net.ssl.keyStore",  keystorePath);
    System.setProperty( "javax.net.ssl.keyStorePassword",keyStorePassword);
    System.setProperty( "javax.net.ssl.keyStoreType", keyStoreType);
    System.setProperty( "javax.net.ssl.trustStore", trustStorePath);
    System.setProperty( "javax.net.ssl.trustStorePassword", trustStorePassword);
    System.setProperty( "javax.net.ssl.trustStoreType", trustStoreType);


    //Datos iniciales del Listener
    String name = "clienteSCAIP";
    String ipAddress = "10.0.2.16";
    String port = "5060";

    String ipAddressProxy = "192.168.1.251";
    String portProxy = "5060";
    String proxyMode = "ON";

    String nameEnd = "servidorSCAIP";
    String ipAddressEnd = "192.168.1.251";
    String portEnd = "5060";

    String transportProtocol = "TLS";
    //Si se usa el protocolo TLS los puertos son 5061
    if(transportProtocol == "TLS"){
        port = "5061";
        portProxy ="5061";
    }
```

```
            //Obtenemos ip del telefono android
            try {
                Context context = getApplicationContext();
                WifiManager wm = (WifiManager) context.getSystemService(Context.WIFI_SERVICE);
                ipAddress = Formatter.formatIpAddress(wm.getConnectionInfo().getIpAddress());
                Log.i("LogMain", "ip = " + ipAddress);

            } catch (Exception e) {
                Log.e("LogMain", "Exception = " + e);
            }

            //Inicializamos el Listener
            SCAIPListener = new SCAIPListener();
            SCAIPListener.init(name, ipAddress, port, ipAddressProxy, portProxy,
transportProtocol, proxyMode, nameEnd, ipAddressEnd, portEnd);
            SCAIPListener.MDBinsert(MDB);
            global.setSCAIPListener(SCAIPListener);


            //Inicializamos el módulo de los mensajes automaticos
            try{
                SCAIPAutoMessager = new SCAIPAutoMessager();
                SCAIPAutoMessager.setGlobal(global);
                SCAIPAutoMessager.MDBinsert(MDB);
                SCAIPAutoMessager.startSendingRequests();
            }catch (Exception e){
                Log.e("MainActivity", "Error: " + e);
            }

            //Nos registramos en el proxy con un REQUEST
            String method = Request.REGISTER;
            Request request = SCAIPListener.createRequest(ipAddressProxy, name, portProxy,
method);
            CallID callIDHeader = (CallID) request.getHeader(CallID.NAME);
            String callID = callIDHeader.getCallId();
            CSeq cSeqHeader = (CSeq) request.getHeader(CSeq.NAME);
            Long cSeq = cSeqHeader.getSeqNumber();
            MDB.insertarRemensaje(callID,cSeq, ipAddressProxy, name, portProxy, method, null);
            Remensaje remensaje = MDB.recuperarRemensajes("callID='" + callID + "'").get(0);
            Log.i("LogMain", "Request : " + request);
            SCAIPListener.MDBinsert(MDB);
            SCAIPListener.globalInsert(global);

            //Thread necesario para enviar mensajes en main thread
            Thread thread = new Thread(new Runnable() {
                @Override
                public void run() {
                    try  {
                        SCAIPListener.sendRequest(request);
                    } catch (Exception e) {
                        Log.e("LogMain", "Exception = " + e);
                    }
                }
            });
            thread.start();

        }catch (Exception e){
            Log.e("LogMain", "Exception = " + e);
        }

    }
    public void GoMenu(View view){
        //Tras el registro nos vamos a la pantalla principal
        Intent intent = new Intent(this, ListaBotones.class);
        startActivity(intent);
    }
}
```

7.2.2 Lista Botones

```
package com.example.sclock.Vistas;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
```

```
import android.widget.Button;

import androidx.recyclerview.widget.RecyclerView;
import androidx.wear.widget.WearableLinearLayoutManager;

import com.example.sclock.R;
import com.example.sclock.basedatos.MiBaseDatos;
import com.example.sclock.basedatos.Plantilla;

import java.util.ArrayList;


public class ListaBotones  extends Activity {

    ArrayList<String> nombresAlarmas;
    private RecyclerView vistaScroll;

    private MiBaseDatos MDB;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.lista_layout);

        MDB = new MiBaseDatos(getApplicationContext());
        //Insertamos las plantillas en la base de datos
        insertarPlantillas();

        //Vamos a crear una vista con scroll
        vistaScroll = (RecyclerView) findViewById(R.id.recyclerid);
        WearableLinearLayoutManager layoutManager = new WearableLinearLayoutManager(this);
        layoutManager.setSmoothScrollbarEnabled(true);
        vistaScroll.setLayoutManager(layoutManager);

        //Recuperamos el nombre de cada tipo de alarmas
        ArrayList<Plantilla> plantillasBaseDatos = MDB.recuperarPlantillas("");
        nombresAlarmas = new ArrayList<String>();
        for (int i = 0; i < plantillasBaseDatos.size(); i++) {
            Plantilla plantilla = plantillasBaseDatos.get(i);
            nombresAlarmas.add(plantilla.getNombreAlarma());
        }

        //Creamos un adaptador personalizado para la vista
        //Hacemos override de un metodo para que al pulsar el boton llame al metodo de alarmas
        AdaptadorRecyclerView adaptadorVistaScroll = new AdaptadorRecyclerView(nombresAlarmas)
        {
            @Override
            public void onEntrada(Object entrada, View view) {
                if (entrada != null) {
                    Button boton = (Button) view.findViewById(R.id.buttonlista);
                    if (boton != null) {
                        String textoAlarma = String.valueOf(boton.getText());
                        boton.setOnClickListener(v -> {
                           GoAlarm(textoAlarma);
                        });
                    }
                }
            }
        };
        vistaScroll.setAdapter(adaptadorVistaScroll);

    }
    public void GoAlarm(String alarmText){
        //Recuperamos la plantilla de la alarma para ver el tipo de alarma
        String type = MDB.recuperarPlantillas("nombreAlarma='" + alarmText +
"'").get(0).getAutoManual();
        //Dependiendo de ella vamos a la vista manual o automatica
        if(type.equals("Manual")){
            Intent intent = new Intent(this, SimpleAlarmView.class);
            Bundle b = new Bundle();
            b.putString("alarmText", alarmText);
            intent.putExtras(b);
            startActivity(intent);
        }else if(type.equals("Automatica")){
            Log.i("ListaBotones", "Tipo 2");
            Intent intent = new Intent(this, ChoiceAlarmView.class);
            Bundle b = new Bundle();
            b.putString("alarmText", alarmText);
```

```
            intent.putExtras(b);
            startActivity(intent);
        }else{
            Log.e("ListaBotones", "Error");
        }

    }

    private void insertarPlantillas(){

        MDB.insertarPlantilla("Help Assistance", "Manual","ref::cid::dty::stc", "stc=001");
        MDB.insertarPlantilla("Pill taken", "Manual", "ref::cid::dty::stc", "stc=001");
        MDB.insertarPlantilla("Heartbeat", "Automatica", "ref::mty::hbo::cid::dty",
"mty=PI::hbo=001");
        MDB.insertarPlantilla("Battery Status","Automatica","ref::cid::dty::stc", "stc=001");
        MDB.insertarPlantilla("GPS Status","Automatica","ref::cid::dty::stc", "stc=001");
        MDB.insertarPlantilla("Temperature Status","Automatica","ref::cid::dty::stc", "stc=001");

    }
}
```

7.2.3 Adaptador Recycler View

```
package com.example.sclock.Vistas;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;

import com.example.sclock.R;

import java.util.ArrayList;

public abstract class AdaptadorRecyclerView extends
RecyclerView.Adapter<AdaptadorRecyclerView.ViewHolderDatos> {

    ArrayList<String> listDatos;
    View view;

    public AdaptadorRecyclerView(ArrayList<String> listDatos) {
        this.listDatos = listDatos;
    }

    @NonNull
    @Override
    public ViewHolderDatos onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        view = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.lista_layout_button,null,false);
        return new ViewHolderDatos(view);
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolderDatos holder, int position) {
        holder.asignarDatos(listDatos.get(position), position);
    }

    @Override
    public int getItemCount() {
        return listDatos.size();
    }

    public abstract void onEntrada(Object entrada, View view);


    public class ViewHolderDatos extends RecyclerView.ViewHolder {

        Button boton;

        public ViewHolderDatos(@NonNull View itemView) {
            super(itemView);
            boton = itemView.findViewById(R.id.buttonlista);
        }

        public void asignarDatos(String dato, int position) {
```

```
            boton.setText(dato);
            onEntrada(boton,view);

        }
    }
}
```

7.2.4 Simple Alarm View

```
package com.example.sclock.Vistas;

import android.app.Activity;
import android.gov.nist.javax.sip.header.CSeq;
import android.gov.nist.javax.sip.header.CallID;
import android.javax.sip.message.Request;
import android.os.Bundle;
import android.util.Log;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.example.sclock.R;
import com.example.sclock.SCAIP.SCAIPConstruct;
import com.example.sclock.SCAIP.SCAIPListener;
import com.example.sclock.SCAIP.SCAIPWebApplication;
import com.example.sclock.basedatos.MiBaseDatos;

public class SimpleAlarmView extends Activity {

    private String nombreAlarma;
    private Button sendButton;
    private Button backButton;
    private TextView textView;
    private SCAIPWebApplication global;
    private SCAIPConstruct SCAIPConstruct;
    private SCAIPListener SCAIPListener;

    private MiBaseDatos MDB;


    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.simple_alarm_layout);

        global = ((SCAIPWebApplication) getApplicationContext());
        SCAIPConstruct = global.getSCAIPConstruct();
        SCAIPListener = global.getSCAIPListener();
        MDB = new MiBaseDatos(getApplicationContext());

        nombreAlarma = getIntent().getStringExtra("alarmText");

        sendButton = (Button) findViewById(R.id.sendButton);
        sendButton.setOnClickListener(v -> {SendAlarm();});
        backButton = (Button) findViewById(R.id.backButton);
        backButton.setOnClickListener(v -> {finish();});

        textView = (TextView) findViewById(R.id.text);
        textView.setText(nombreAlarma);
    }

    public void SendAlarm(){
        //Construimos el mensaje SCAIP
        String XMLMessage = SCAIPConstruct.CreateRequest(nombreAlarma);
        Request request = SCAIPListener.createRequestEnd(XMLMessage,Request.MESSAGE);

        //Guadamos el mensaje para un posible reenvio
        CallID callIDHeader = (CallID) request.getHeader(CallID.NAME);
        String callID = callIDHeader.getCallId();
        CSeq cSeqHeader = (CSeq) request.getHeader(CSeq.NAME);
        Long cSeg = cSeqHeader.getSeqNumber();
        MDB.insertarRemensaje(callID,cSeg,null,null,null,Request.MESSAGE,XMLMessage);
        Log.i("SimpleAlarmView", "Request : " + request);

        //Creamos un hilo debido a que no puede ser enviado un mensaje en el main thread
        Thread thread = new Thread(new Runnable() {
            @Override
```

```
            public void run() {
                try  {
                    SCAIPListener.sendRequest(request);
                } catch (Exception e) {
                    Log.e("LogMain", "Exception = " + e);
                }
            }
        });
        thread.start();
        Toast.makeText(getApplicationContext(), "Alarma Enviada", Toast.LENGTH_LONG).show();
        //Despues de enviar la alarma vuelve al menu anterior
        finish();
    }
}
```

7.2.5 Choice Alarm View

```
package com.example.sclock.Vistas;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.example.sclock.R;
import com.example.sclock.basedatos.MiBaseDatos;
import com.example.sclock.basedatos.Plantilla;

public class ChoiceAlarmView extends Activity {

    private String nombreAlarma;
    private Button firstButton;
    private Button secondButton;
    private Button thirdButton;
    private Button fourthButton;
    private Button fifthButton;
    private Button backButton;
    private TextView textView;

    private MiBaseDatos MDB;


    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.choice_alarm_layout);
        MDB = new MiBaseDatos(getApplicationContext());

        nombreAlarma = getIntent().getStringExtra("alarmText");

        firstButton = (Button) findViewById(R.id.firstButton);
        firstButton.setOnClickListener(v -> {SendAlarm("1");});
        secondButton = (Button) findViewById(R.id.secondButton);
        secondButton.setOnClickListener(v -> {SendAlarm( "2");});
        thirdButton = (Button) findViewById(R.id.thirdButton);
        thirdButton.setOnClickListener(v -> {SendAlarm("5");});
        fourthButton = (Button) findViewById(R.id.fourthButton);
        fourthButton.setOnClickListener(v -> {SendAlarm("10");});
        fifthButton = (Button) findViewById(R.id.fifthButton);
        fifthButton.setOnClickListener(v -> {SendAlarm("0");});

        backButton = (Button) findViewById(R.id.backButton);
        backButton.setOnClickListener(v -> {finish();});
        textView = (TextView) findViewById(R.id.text);
        textView.setText(nombreAlarma);
    }

    public void SendAlarm(String botonPulsado){
        //Cargamos la alarma o la borramos
        if(botonPulsado.equals("0")){
            try{
                MDB.borrarAutomatica(nombreAlarma);
                Log.i("SimpleAlarmView", "Alarma Borrada");
                Toast.makeText(getApplicationContext(),"Alarma Borrada",
Toast.LENGTH_LONG).show();
            }catch (Exception e){
```

```
                Log.e("ChoiceAlarm","Error: " + e);
            }
        }
        else{
            try {
                Plantilla plantilla = MDB.recuperarPlantillas("nombreAlarma='" + nombreAlarma +
"'").get(0);
                MDB.insertarAutomatica(nombreAlarma, plantilla.getArgumentosSet(), botonPulsado);
                Log.i("SimpleAlarmView", "Alarma Programada");
                Toast.makeText(getApplicationContext(), "Alarma Programada",
Toast.LENGTH_LONG).show();
            }catch (Exception e){
                Log.e("ChoiceAlarm","Error: " + e);
            }
        }
        //Despues de enviar la alarma vuelve al menu anterior
        finish();

    }
}
```

7.2.6 SCAIP Listener

```
package com.example.sclock.SCAIP;

import android.gov.nist.javax.sip.header.CSeq;
import android.gov.nist.javax.sip.header.CallID;
import android.javax.sip.ClientTransaction;
import android.javax.sip.DialogTerminatedEvent;
import android.javax.sip.IOExceptionEvent;
import android.javax.sip.ListeningPoint;
import android.javax.sip.RequestEvent;
import android.javax.sip.ResponseEvent;
import android.javax.sip.ServerTransaction;
import android.javax.sip.SipFactory;
import android.javax.sip.SipListener;
import android.javax.sip.SipProvider;
import android.javax.sip.SipStack;
import android.javax.sip.TimeoutEvent;
import android.javax.sip.TransactionTerminatedEvent;
import android.javax.sip.address.Address;
import android.javax.sip.address.AddressFactory;
import android.javax.sip.address.SipURI;
import android.javax.sip.address.URI;
import android.javax.sip.header.AuthorizationHeader;
import android.javax.sip.header.CSeqHeader;
import android.javax.sip.header.CallIdHeader;
import android.javax.sip.header.ContactHeader;
import android.javax.sip.header.ContentTypeHeader;
import android.javax.sip.header.FromHeader;
import android.javax.sip.header.HeaderFactory;
import android.javax.sip.header.MaxForwardsHeader;
import android.javax.sip.header.ToHeader;
import android.javax.sip.header.ViaHeader;
import android.javax.sip.header.WWWAuthenticateHeader;
import android.javax.sip.message.MessageFactory;
import android.javax.sip.message.Request;
import android.javax.sip.message.Response;
import android.util.Log;

import com.example.sclock.basedatos.MiBaseDatos;
import com.example.sclock.basedatos.Remensaje;

import org.apache.commons.codec.digest.DigestUtils;

import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Properties;

public class SCAIPListener implements SipListener {

    private static SipProvider sipProvider;
    private static AddressFactory addressFactory;
    private static MessageFactory messageFactory;
    private static HeaderFactory headerFactory;
```

```
    private static SipStack sipStack;
    private ListeningPoint listeningPoint;
    private ClientTransaction inviteTid;
    private ContactHeader contactHeader;


    private String name;
    private String ipAddress;
    private String port;
    private String ipAddressProxy;
    private String portProxy;
    private String transportProtocol;
    private String proxyMode;
    private String nameEnd;
    private String ipAddressEnd;
    private String portEnd;


    private MiBaseDatos MDB;
    private SCAIPWebApplication global;

    public void MDBinsert(MiBaseDatos MDB){
        this.MDB=MDB;
    }
    public void globalInsert(SCAIPWebApplication global){this.global = global;}

    //Inicializamos el Listener (Revisar protocolos transporte)
    public void init(String name, String ipAddress, String port,
                     String ipAddressProxy, String portProxy, String transportProtocol, String
proxyMode,
                     String nameEnd, String ipAddressEnd, String portEnd){
        this.name = name;
        this.ipAddress = ipAddress;
        this.port = port;
        this.ipAddressProxy = ipAddressProxy;
        this.portProxy = portProxy;
        this.transportProtocol = transportProtocol;
        this.proxyMode = proxyMode;
        this.nameEnd = nameEnd;
        this.ipAddressEnd = ipAddressEnd;
        this.portEnd = portEnd;

        //Inicialización de la factoria
        SipFactory sipFactory = null;
        sipStack = null;
        sipFactory = SipFactory.getInstance();
        sipFactory.setPathName("android.gov.nist"); //Este path determina donde estan las
implementaciones de la libreria

        //Declaracion de las propiedades
        Properties properties = new Properties();

        //Declaramos el router para los mensajes sip
        if(proxyMode.equals("ON")) {
            properties.setProperty("android.javax.sip.OUTBOUND_PROXY", ipAddressProxy + ":" +
portProxy + "/" + transportProtocol);
        }
        properties.setProperty("android.javax.sip.STACK_NAME", name);

properties.setProperty("android.gov.nist.javax.sip.ENABLED_CIPHER_SUITES","TLS_RSA_WITH_AES_128_C
BC_SHA");
        properties.setProperty("android.gov.nist.javax.sip.TLS_CLIENT_AUTH_TYPE", "DisabledAll");

        try {
            //Creamos el stack
            sipStack = sipFactory.createSipStack(properties);

            //Inicializamos las factorias necesarias
            headerFactory = sipFactory.createHeaderFactory();
            addressFactory = sipFactory.createAddressFactory();
            messageFactory = sipFactory.createMessageFactory();

            //Creamos un Listening Point
            listeningPoint = sipStack.createListeningPoint(ipAddress, Integer.parseInt(port),
transportProtocol);

            //Creamos un sip Provider
            sipProvider = sipStack.createSipProvider(listeningPoint);
```

```
            sipProvider.addSipListener(this);
        } catch (Exception e) {
            Log.e("LogListener", "Error: " + e);
        }
    }

    public Request createRequest(String toSipAddress, String toUser, String toSipPort, String
requestMethod) {
        Request request = null;

        try {
            // Crea cabecera From
            SipURI fromAddress = addressFactory.createSipURI(name, ipAddress);
            Address fromNameAddress = addressFactory.createAddress(fromAddress);
            fromNameAddress.setDisplayName(name);
            FromHeader fromHeader = headerFactory.createFromHeader(fromNameAddress, "12345");

            // Crea cabecera To
            SipURI toAddress = addressFactory.createSipURI(toUser, toSipAddress);
            Address toNameAddress = addressFactory.createAddress(toAddress);
            toNameAddress.setDisplayName(toUser);
            ToHeader toHeader = headerFactory.createToHeader(toNameAddress, null);

            // Crea RequestURI
            SipURI requestURI = addressFactory.createSipURI(toUser, toSipAddress + ":" +
toSipPort);

            // Crea ViaHeaders
            ArrayList viaHeaders = new ArrayList();
            String ipAddress = listeningPoint.getIPAddress();
            String transport = listeningPoint.getTransport();
            ViaHeader viaHeader = headerFactory.createViaHeader(ipAddress,
sipProvider.getListeningPoint(transport).getPort(), transport, null);
            viaHeaders.add(viaHeader);

            // Max Forwards Headers
            MaxForwardsHeader maxForwards = headerFactory.createMaxForwardsHeader(70);
            CallIdHeader callIdHeader = sipProvider.getNewCallId();
            CSeqHeader cSeqHeader = headerFactory.createCSeqHeader(1L, requestMethod);

            // Crea Request
            request = messageFactory.createRequest(requestURI, requestMethod, callIdHeader,
cSeqHeader, fromHeader, toHeader, viaHeaders, maxForwards);

            // Crea cabecera contacto
            SipURI contactURI = addressFactory.createSipURI(name, ipAddress);
            contactURI.setPort(sipProvider.getListeningPoint(transport).getPort());
            contactURI.setTransportParam(transport);
            Address contactAddress = addressFactory.createAddress(contactURI);
            contactAddress.setDisplayName(name);
            contactHeader = headerFactory.createContactHeader(contactAddress);
            request.addHeader(contactHeader);

        }catch (Exception e){
            Log.e("LogListener", name + ": Exception = " + e);
        }
        return request;
    }

    public Request createRequest(String toSipAddress, String toUser, String toSipPort, String
requestMethod, String dataMessage){
        Request request = this.createRequest(toSipAddress,toUser,toSipPort,requestMethod);
        try {
            //Deberia ser application, scaip+xml
            ContentTypeHeader contentTypeHeader = headerFactory.createContentTypeHeader("text",
"plain");
            request.setContent(dataMessage, contentTypeHeader);
        }catch(Exception e){
            Log.e("LogListener", name + ": Exception = " + e);
        }
        return request;
    }

    public Request createRequestProxy(String message, String method){
        Request request = null;
        if(!message.equals(null)) {
            request = this.createRequest(ipAddressProxy, name, portProxy, method, message);
```

```
        }else {
            request = this.createRequest(ipAddressProxy, name, portProxy, method);
        }
        return request;
    }

    public Request createRequestEnd(String message, String method){
        Request request = null;
        if(!message.equals(null)) {
            request = this.createRequest(ipAddressEnd, nameEnd, portEnd, method, message);
        }else {
            request = this.createRequest(ipAddressEnd, nameEnd, portEnd, method);
        }
        return request;
    }


    public void sendRequest(Request request){
        try {
            CallID callIDHeader = (CallID) request.getHeader(CallID.NAME);
            String callID = callIDHeader.getCallId();

            //Crea la transaccion y envia la request
            inviteTid = sipProvider.getNewClientTransaction(request);
            inviteTid.sendRequest();
        }catch (Exception e){
            Log.e("LogListener", name + ": Exception = " + e);
        }
    }

    @Override
    public void processRequest(RequestEvent requestEvent) {

        Request request = requestEvent.getRequest();
        Log.i("LogListener","Request: " + request);
        ServerTransaction serverTransactionId = requestEvent.getServerTransaction();

        if (request.getMethod().equals(Request.INVITE)) {
            processInvite(requestEvent, serverTransactionId);
        } else if (request.getMethod().equals(Request.ACK)) {
            processAck(requestEvent, serverTransactionId);
        } else if (request.getMethod().equals(Request.BYE)) {
            processBye(requestEvent, serverTransactionId);
        } else if (request.getMethod().equals(Request.CANCEL)) {
            processCancel(requestEvent, serverTransactionId);
        } else if (request.getMethod().equals(Request.MESSAGE)) {
            processMessage(requestEvent, serverTransactionId);
        } else if (request.getMethod().equals(Request.OPTIONS)) {
            processOptions(requestEvent, serverTransactionId);
        } else {
            Log.e("LogListener", "Error en la transaccion NO RESPETA METODOS");
        }

    }

    public void processInvite(RequestEvent requestEvent, ServerTransaction serverTransaction) {
    }

    public void processAck(RequestEvent requestEvent, ServerTransaction serverTransaction){
        Log.i("LogListener", "LLEGO UN ACK QUE?????");
    }

    public void processBye(RequestEvent requestEvent, ServerTransaction serverTransaction){}

    public void processCancel(RequestEvent requestEvent, ServerTransaction serverTransaction){}

    public void processMessage(RequestEvent requestEvent, ServerTransaction serverTransaction) {
        SipProvider sipProvider = (SipProvider) requestEvent.getSource();
        Request request = requestEvent.getRequest();
        try {
            Log.i("LogListener", name + " : Procesando Message");
            ServerTransaction st = requestEvent.getServerTransaction();

            if (st == null) {
                st = sipProvider.getNewServerTransaction(request);
            }

            Response responseOK = messageFactory.createResponse(Response.OK, request);
```

```
            /*  *//*SipURI contactURI = addressFactory.createSipURI(name, ipAddress);
              contactURI.setPort(sipProvider.getListeningPoint(transportProtocol).getPort());
              Address contactAddress = addressFactory.createAddress(contactURI);
              contactAddress.setDisplayName(name);
              contactHeader = headerFactory.createContactHeader(contactAddress);*//*
              responseOK.addHeader(contactHeader);*/

              st.sendResponse(responseOK);
              Log.i("LogListener", name + ": Response = " + responseOK);
          } catch (Exception e) {
              Log.e("LogListener", name + " : Error : " + e);
          }
          try {
              ContentTypeHeader contentTypeHeader = (ContentTypeHeader)
request.getHeader(ContentTypeHeader.NAME);
              if(!contentTypeHeader.equals(null)){
                  Log.i("LogL","SCAIP");
                  byte[] content = request.getRawContent();
                  String body = new String(content, StandardCharsets.UTF_8);
                  if(body.startsWith("<mrq>")) {
                      String ref1 = body.split("<ref>")[1];
                      String ref = ref1.split("</ref>")[0];
                      HashMap<String, String> map = new HashMap<String, String>();
                      map.put("ref", ref);
                      String SCAIPString = global.getSCAIPConstruct().CreateResponse(map);
                      Request SCAIPResponse = createRequestEnd(SCAIPString, Request.MESSAGE);
                      CallID callIDHeader = (CallID) SCAIPResponse.getHeader(CallID.NAME);
                      String callID = callIDHeader.getCallId();
                      CSeq cSeqHeader = (CSeq) request.getHeader(CSeq.NAME);
                      Long cSeq = cSeqHeader.getSeqNumber();
                      Boolean bool = MDB.insertarRemensaje(callID, cSeq, ipAddressEnd, nameEnd,
portEnd, Request.MESSAGE, SCAIPString);
                      Log.i("LogL", "Request:" + SCAIPResponse);
                      sendRequest(SCAIPResponse);
                  }
              }
          }catch (Exception e) {
              Log.e("LogListener", name + " : Error : " + e);
          }


      }


      @Override
      public void processResponse(ResponseEvent responseReceivedEvent) {

          Response response = (Response) responseReceivedEvent.getResponse();
          Log.i("LogListener", "Response = " + response);

          try{
              //Necesitamos autenticarnos en el servidor
              if(response.getStatusCode() == Response.UNAUTHORIZED){
                  // Obtenemos el encabezado de autenticación
                  WWWAuthenticateHeader wwwAuthHeader = null;
                  wwwAuthHeader= (WWWAuthenticateHeader)
response.getHeader(WWWAuthenticateHeader.NAME);
                  if( wwwAuthHeader != null){
                      // Obtenemos los parámetros necesarios para el cálculo de la respuesta
                      String username = name;
                      String realm = wwwAuthHeader.getRealm();
                      String nonce = wwwAuthHeader.getNonce();
                      String password = "s1234"; // Reemplaza con tu contraseña

                      ToHeader toHeader = (ToHeader) response.getHeader(ToHeader.NAME);
                      URI serverURI = toHeader.getAddress().getURI();

                      CSeq cSeqHeader = (CSeq) response.getHeader(CSeq.NAME);
                      Long cSeq = cSeqHeader.getSeqNumber();
                      String method = cSeqHeader.getMethod();


                      //Formula para el calculo de la respuesta de autenticacion
                      String a1 = username + ":" + realm + ":" + password;
                      String a2 = method + ":" + serverURI;
                      String responseValue = DigestUtils.md5Hex(DigestUtils.md5Hex(a1) + ":" +
```

```
nonce + ":" + DigestUtils.md5Hex(a2));

                    AuthorizationHeader authHeader =
headerFactory.createAuthorizationHeader("Digest");
                    authHeader.setUsername(username);
                    authHeader.setRealm(realm);
                    authHeader.setNonce(nonce);
                    authHeader.setURI(serverURI);
                    authHeader.setResponse(responseValue);
                    authHeader.setAlgorithm("MD5");

                    //Recuperamos el mensaje que ha sido denegado para reenviarlo
                    CallID callIDHeader = (CallID) response.getHeader(CallID.NAME);
                    String callID = callIDHeader.getCallId();
                    Remensaje remensaje = MDB.recuperarRemensajes("callID='" + callID +
"'").get(0);
                    Request request = null;
                    if(remensaje.getIpAddress()!=null){
                        request = this.createRequest(remensaje.getIpAddress(),
remensaje.getName(),remensaje.getPort(),remensaje.getMethod(),remensaje.getMessage());
                    }else {
                        request = this.createRequestEnd(remensaje.getMessage(),
remensaje.getMethod());
                    }
                    // Agrega el encabezado de autorización a la solicitud
                    request.addHeader(authHeader);
                    request.setHeader(callIDHeader);
                    CSeqHeader newCSeqHeader = headerFactory.createCSeqHeader(cSeq +1L,
remensaje.getMethod());
                    request.setHeader(newCSeqHeader);
                    Log.i("LogListener", "Request :"+ request );
                    sendRequest(request);
                }
            }
        }
        catch (Exception e){
            Log.i("sd","sds");
            Log.e("LogListener", String.valueOf(e));
        }
    }

    //El mensaje para mostrar que estamos vivos
    public void processOptions(RequestEvent requestEvent, ServerTransaction serverTransaction){
        SipProvider sipProvider = (SipProvider) requestEvent.getSource();
        Request request = requestEvent.getRequest();
        try {
            ServerTransaction st = requestEvent.getServerTransaction();
            if (st == null) {
                st = sipProvider.getNewServerTransaction(request);
            }

            Response responseOK = messageFactory.createResponse(Response.OK, request);
            st.sendResponse(responseOK);
            Log.i("LogListener", "Response = " + responseOK);
        } catch (Exception e) {
            Log.e("LogListener","Error : " + e);
        }
    }

    @Override
    public void processTimeout(TimeoutEvent timeoutEvent) {
    }

    //No me importa nada de abajo
    public void processIOException(IOExceptionEvent exceptionEvent) {
        Log.e("LogListener","IOException happened for "
                + exceptionEvent.getHost() + " port = "
                + exceptionEvent.getPort());
    }

    public void processTransactionTerminated(
            TransactionTerminatedEvent transactionTerminatedEvent) {
        Log.e("LogListener","Transaction terminated event recieved");
    }

    public void processDialogTerminated(
            DialogTerminatedEvent dialogTerminatedEvent) {
        Log.e("LogListener","Dialog Terminated Event");
```

```
    }


}
```

7.2.7 SCAIP Construct

```
package com.example.sclock.SCAIP;

import android.util.Log;
import android.util.Xml;

import com.example.sclock.basedatos.*;
import org.xmlpull.v1.XmlSerializer;

import java.io.StringReader;
import java.io.StringWriter;
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Random;

import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.Schema;
import javax.xml.validation.Validator;


public class SCAIPConstruct {


    private String reference;
    private String version;
    private String controller_id;
    private String device_type;

    private String type;
    private Schema schema;
    private MiBaseDatos MDB;

    private String request;
    ArrayList<String> ordenMrq = new ArrayList<String>(Arrays.asList(

"ref","ver","sco","cha","mty","hbo","cid","dty","did","dco","dte","crd","stc","stt","pri","lco","
lva","lge","lte","ico","ite","ame"));
    private String response;
    ArrayList<String> ordenMrs = new ArrayList<String>(Arrays.asList(
            "ref","snu","ste","cve","mre","cre","tnu","hbi"));



    public void MDBinsert(MiBaseDatos MDB){
        this.MDB=MDB;
    }

    public SCAIPConstruct(String version, String controller_id, String device_type, Schema
schema){
        this.version=version;
        this.controller_id=controller_id;
        this.device_type=device_type;
        this.schema=schema;
    }

    public String CreateRequest(String type){

        HashMap<String,String> argumentos = new HashMap<String,String>();
        request = CreateRequest(type,argumentos);
        return request;
    }

    public String CreateRequest(String type, HashMap arguments){


        try{
```

```
            this.type = type;
            Plantilla plantilla = MDB.recuperarPlantillas("nombreAlarma='" + type + "'").get(0);
            ArrayList<String> argumentosPlantilla = MDB.stringDecod(plantilla.getArgumentos());
            ArrayList<String> argumentosSetPlantilla =
MDB.stringDecod(plantilla.getArgumentosSet());
            for(String argumento: argumentosSetPlantilla) {
                String[] partes = argumento.split("=");
                arguments.put(partes[0],partes[1]);
            }

            for(String argumento: argumentosPlantilla){
                if(arguments.get(argumento)==null) {
                    try {
                        arguments.put(argumento, argumentoGeneral(argumento));
                    } catch (Exception ex) {
                        Log.e("SCAIPConstruct", String.valueOf(ex));
                    }
                }
            }
            request = CreateXML(arguments, ordenMrq,"mrq");

        }catch(Exception e){
            Log.e("SCAIPConstruct", String.valueOf(e));
        }
        return request;
    }

    public String CreateXML(HashMap arguments, ArrayList<String> order, String type){


        String XML = null;
        //Construye en orden el XML SCAIP con los argumentos recibidos
        try {
            XmlSerializer serializer = Xml.newSerializer();
            StringWriter writer = new StringWriter();
            serializer.setOutput(writer);
            if(type.equals("mrq")) {
                serializer.startTag(null, "mrq");
            }else if(type.equals("mrs")){
                serializer.startTag(null, "mrs");
            }
            for (String value: order) {
                if (value == "lge" && arguments.get(value)!=null) {
                    serializer.startTag(null, "lge");

                    serializer.startTag(null, "geo");
                    serializer.text((String) arguments.get("geo"));
                    serializer.endTag(null, "geo");

                    serializer.startTag(null, "tim");
                    serializer.text((String) arguments.get("tim"));
                    serializer.endTag(null, "tim");

                    serializer.startTag(null, "gga");
                    serializer.text((String) arguments.get("gga"));
                    serializer.endTag(null, "gga");

                    serializer.endTag(null, "lge");
                } else if(arguments.get(value)!=null){
                    serializer.startTag(null, value);
                    serializer.text((String) arguments.get(value));
                    serializer.endTag(null, value);
                }
            }

            if(type.equals("mrq")) {
                serializer.endTag(null, "mrq");
            }else if(type.equals("mrs")){
                serializer.endTag(null, "mrs");
            }

            serializer.endDocument();

            XML = writer.toString();
            Log.d("XML", XML);
        } catch (Exception e) {
            Log.e("SCAIPConstruct", String.valueOf(e));
        }
```

```
        Boolean bool = SCAIPConstruct.validarXML(XML,schema);
        Log.i("SCAIPConstruct", String.valueOf(bool));

        return XML;
    }

    public String argumentoGeneral(String argumento){

        String argumentoGeneral = null;
        switch (argumento){
            case "ref":
                Random random = new Random();
                int numeroAleatorio = random.nextInt(900) + 100;
                reference = type.substring(0,3) + numeroAleatorio;
                argumentoGeneral = reference;
                break;
            case "ver" :
                argumentoGeneral = version;
                break;
            case "cid":
                argumentoGeneral = controller id;
                break;
            case "dty":
                argumentoGeneral = device_type;
                break;
            default:
                Log.e("Error", "Error");
        }
        return argumentoGeneral;
    }


    public static boolean validarXML(String xmlString, Schema schema) {

        // Valida que el XML concuerda con el schema SCAIP
        try {
            Source xmlSource = new StreamSource(new StringReader(xmlString));
            Validator validator = schema.newValidator();
            validator.validate(xmlSource);
        } catch (Exception e) {
            Log.e("Validator", String.valueOf(e));
            return false;
        }

        return true;
    }

    public String CreateResponse(HashMap arguments){


        try{
            arguments.put("snu", "0");
            response = CreateXML(arguments, ordenMrs,"mrs");

        }catch(Exception e){
            Log.e("SCAIPConstruct", String.valueOf(e));
        }
        return response;
    }

}
```

7.2.8 SCAIP Auto Messager

```
package com.example.sclock.SCAIP;

import android.gov.nist.javax.sip.header.CSeq;
import android.gov.nist.javax.sip.header.CallID;
import android.javax.sip.message.Request;
import android.util.Log;

import com.example.sclock.basedatos.Automatica;
import com.example.sclock.basedatos.MiBaseDatos;

import java.util.ArrayList;
import java.util.HashMap;
```

```
public class SCAIPAutoMessager {

    private static final int DELAY = 15000; // Delay inicial en milisegundos
    private static final int PERIOD = 40000; // Periodo en milisegundos

    private Thread requestThread;
    private boolean isRunning;

    private SCAIPWebApplication global;
    private MiBaseDatos MDB;
    private SCAIPConstruct SCAIPConstruct;
    private SCAIPListener SCAIPListener;

    private ArrayList<Automatica> lista_automaticas;
    private String nombreAlarma;
    private String argumentosValor;
    private String tiempo;
    private int timerCount = 1;

    public void MDBinsert(MiBaseDatos MDB){
        this.MDB=MDB;
    }
    public void setGlobal(SCAIPWebApplication global){
        this.global=global;
    }

    public void startSendingRequests() {
        if (requestThread != null && requestThread.isAlive()) {
            // El hilo ya está en ejecución, no se inicia nuevamente
            return;
        }

        isRunning = true;
        SCAIPConstruct = global.getSCAIPConstruct();
        SCAIPListener = global.getSCAIPListener();


        requestThread = new Thread(new Runnable() {
            @Override
            public void run() {

                //Ponemos un delay desde que se inicia
                try {
                    Thread.sleep(DELAY);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }

                //Hilo eterno de envio de mensajes
                while (isRunning) {
                    try {
                        //Recupera toods los mensajes automaticos
                        Log.i("SCAIPAutoMessager", SCAIPListener.toString());
                        lista_automaticas = MDB.recuperarAutomaticas("");
                        //Por cada mensaje automatico mira el periodo y lo envia si procede
                        for (Automatica mensaje : lista_automaticas) {
                            nombreAlarma = mensaje.getNombreAlarma();
                            argumentosValor = mensaje.getArgumentosValor();
                            tiempo = mensaje.getTiempo();
                            switch (tiempo) {
                                case "1":
                                    logicaDeEnvio();
                                    break;
                                case "2":
                                    if (timerCount % 2 == 0) {
                                        Log.e("SCAIPAutoMessager", "2");
                                        logicaDeEnvio();
                                    }
                                    break;
                                case "5":
                                    if (timerCount % 5 == 0) {
                                        Log.e("SCAIPAutoMessager", "5");
                                        logicaDeEnvio();
                                    }
                                    break;
                                case "10":
                                    if (timerCount % 10 == 0) {
```

```
                                    Log.e("SCAIPAutoMessager", "10");
                                    logicaDeEnvio();
                                }
                                break;
                            default:
                                Log.e("SCAIPAutoMessager", "Errocito");
                                break;
                        }

                    }

                    try {
                        Thread.sleep(PERIOD);
                        timerCount++;
                        if(timerCount==11){
                            timerCount=1;
                        }
                    } catch (InterruptedException e) {
                        Log.e("SCAIPAutoMessager", String.valueOf(e));
                    }
                }catch (Exception e){
                    Log.i("SCAIPAutoMessager", "Estoy vivo" + e);
                    try {
                        Thread.sleep(PERIOD);
                    } catch (InterruptedException ex) {
                        Log.e("SCAIPAutoMessager", String.valueOf(e));
                    }
                }
            }
        }
    });

    requestThread.start();
}

public void stopSendingRequests() {
    isRunning = false;
    if (requestThread != null && requestThread.isAlive()) {
        requestThread.interrupt();
        requestThread = null;
    }
}

public void logicaDeEnvio(){
    //Divide los argumentos y sus valores
    String[] argval = argumentosValor.split("::");
    HashMap<String, String> mapa = new HashMap<String, String>();
    for (String args : argval) {
        String[] splitargs = args.split("=");
        mapa.put(splitargs[0], splitargs[1]);
    }
    //Construye el mensaje SCAIP
    String XMLMessage = SCAIPConstruct.CreateRequest(nombreAlarma, mapa);
    Request request = SCAIPListener.createRequestEnd(XMLMessage, Request.MESSAGE);
    CallID callIDHeader = (CallID) request.getHeader(CallID.NAME);
    String callID = callIDHeader.getCallId();
    CSeq cSeqHeader = (CSeq) request.getHeader(CSeq.NAME);
    Long cSeq = cSeqHeader.getSeqNumber();
    //Guarda el mensaje por un posible reenvio y lo envia
    MDB.insertarRemensaje(callID,cSeq,null,null,null,Request.MESSAGE,XMLMessage);
    SCAIPListener.sendRequest(request);
    Log.i("AutoMessager", "Request: " + request);
}
}
```

7.2.9 SCAIP Web Application

```
package com.example.sclock.SCAIP;

import android.app.Application;

public class SCAIPWebApplication extends Application{

    private static SCAIPWebApplication application;
    private SCAIPListener SCAIPListener;
    private SCAIPConstruct SCAIPConstruct;
```

```
    public void onCreate() {
        super.onCreate();
        application = this;
    }
    public synchronized static SCAIPWebApplication getInstance() { return application; }

    public void setSCAIPListener (SCAIPListener SCAIPListener) { this.SCAIPListener =
SCAIPListener; }
    public SCAIPListener getSCAIPListener() {return SCAIPListener;}

    public void setSCAIPConstruct (SCAIPConstruct SCAIPConstruct) { this.SCAIPConstruct =
SCAIPConstruct; }
    public SCAIPConstruct getSCAIPConstruct() {return SCAIPConstruct;}
}
```

7.2.10 MiBaseDatos

```
package com.example.sclock.basedatos;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.StringJoiner;

public class MiBaseDatos extends SQLiteOpenHelper {


    private static final int VERSION_BASEDATOS = 1;
    private static final String NOMBRE_BASEDATOS = "scaippa.db";
    private static final String TABLA_PLANTILLAS ="CREATE TABLE IF NOT EXISTS plantillas " +
            "(nombreAlarma STRING, autoManual STRING, argumentos STRING, argumentosSet STRING,
PRIMARY KEY(nombreAlarma))";
    private static final String TABLA_AUTOMATICA ="CREATE TABLE IF NOT EXISTS automatica " +
            "(nombreAlarma STRING, argumentosValor STRING, tiempo STRING, PRIMARY
KEY(nombreAlarma))";
    private static final String TABLA_REMENSAJES ="CREATE TABLE IF NOT EXISTS remensajes " +
            "(callID STRING, cSeq LONG, ipAddress STRING, name STRING, port STRING, method
STRING, message STRING, PRIMARY KEY(callid, cSeq))";


    public MiBaseDatos(Context context) {
        super(context, NOMBRE_BASEDATOS, null, VERSION_BASEDATOS);
    }

    @Override
    public void onCreate(SQLiteDatabase db)
    {
        db.execSQL(TABLA_PLANTILLAS);
        db.execSQL(TABLA_AUTOMATICA);
        db.execSQL(TABLA_REMENSAJES);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS plantillas"  + TABLA_PLANTILLAS);
        db.execSQL("DROP TABLE IF EXISTS automatica"  + TABLA_AUTOMATICA);
        db.execSQL("DROP TABLE IF EXISTS remensajes"  + TABLA_REMENSAJES);
        onCreate(db);
    }

    //Funciones tabla plantillas
    public boolean insertarPlantilla(String nombreAlarma, String autoManual, String argumentos,
String argumentosSet) {
        long salida=0;
        SQLiteDatabase db = getWritableDatabase();
        if (db != null) {
            ContentValues valores = new ContentValues();
            valores.put("nombreAlarma", nombreAlarma);
            valores.put("autoManual", autoManual);
            valores.put("argumentos", argumentos);
```

```
            valores.put("argumentosSet", argumentosSet);
            salida=db.insert("plantillas", null, valores);
        }
        db.close();
        return(salida>0);
    }

    public boolean borrarPlantillas() {
        SQLiteDatabase db = getWritableDatabase();
        long salida=0;
        if (db != null) {
            salida=db.delete("plantillas", null, null);
        }
        db.close();
        return(salida>0);
    }

    public ArrayList<Plantilla> recuperarPlantillas(String seleccion) {
        SQLiteDatabase db = getReadableDatabase();
        ArrayList<Plantilla> lista_plantillas = new ArrayList<Plantilla>();
        String[] valores recuperar = {"nombreAlarma", "autoManual", "argumentos",
"argumentosSet"};
        Cursor c = db.query("plantillas", valores_recuperar, seleccion, null, null, null,
"nombreAlarma DESC", null);
        c.moveToFirst();
        do {
            Plantilla plantilla = new Plantilla(c.getString(0), c.getString(1), c.getString(2),
c.getString(3));
            lista_plantillas.add(plantilla);
        } while (c.moveToNext());
        db.close();
        c.close();
        return lista plantillas;
    }



    //Funciones tabla automatica
    public boolean insertarAutomatica(String nombreAlarma, String argumentosValor, String tiempo)
{
        long salida=0;
        SQLiteDatabase db = getWritableDatabase();
        if (db != null) {
            ContentValues valores = new ContentValues();
            valores.put("nombreAlarma", nombreAlarma);
            valores.put("argumentosValor", argumentosValor);
            valores.put("tiempo", tiempo);
            salida=db.insert("automatica", null, valores);
        }
        db.close();
        return(salida>0);
    }

    public boolean borrarAutomatica(String nombreAlarma) {
        SQLiteDatabase db = getWritableDatabase();
        long salida=0;
        if (db != null) {
            salida=db.delete("automatica", "nombreAlarma='" + nombreAlarma + "'", null);
        }
        db.close();
        return(salida>0);
    }

    public boolean borrarAutomaticas() {
        SQLiteDatabase db = getWritableDatabase();
        long salida=0;
        if (db != null) {
            salida=db.delete("automatica", null, null);
        }
        db.close();
        return(salida>0);
    }

    public ArrayList<Automatica> recuperarAutomaticas(String seleccion) {
        SQLiteDatabase db = getReadableDatabase();
        ArrayList<Automatica> lista_automatica = new ArrayList<Automatica>();
```

```
        String[] valores_recuperar = {"nombreAlarma", "argumentosValor", "tiempo"};
        Cursor c = db.query("automatica", valores_recuperar, seleccion, null, null, null,
"nombreAlarma DESC", null);
        c.moveToFirst();
        do {
            Automatica automatica = new Automatica(c.getString(0), c.getString(1),
c.getString(2));
            lista_automatica.add(automatica);
        } while (c.moveToNext());
        db.close();
        c.close();
        return lista_automatica;
    }

    //tabla remensajes
    public boolean insertarRemensaje(String callID, Long cSeq, String ipAddress, String name,
String port, String method, String message) {
        long salida = 0;
        try {
            SQLiteDatabase db = getWritableDatabase();
            if (db != null) {
                ContentValues valores = new ContentValues();
                valores.put("callID", callID);
                valores.put("cSeq", cSeq);
                valores.put("ipAddress", ipAddress);
                valores.put("name", name);
                valores.put("port", port);
                valores.put("method", method);
                valores.put("message", message);
                salida = db.insert("remensajes", null, valores);
            }
            db.close();
        }catch (Exception e){
            Log.e("Log", "Error: " + e);
        }
        return(salida>0);
    }

    public boolean borrarRemensajes() {
        SQLiteDatabase db = getWritableDatabase();
        long salida=0;
        if (db != null) {
            salida=db.delete("remensajes", null, null);
        }
        db.close();
        return(salida>0);
    }

    public ArrayList<Remensaje> recuperarRemensajes(String seleccion) {
        SQLiteDatabase db = getReadableDatabase();
        ArrayList<Remensaje> lista_remensajes = new ArrayList<Remensaje>();
        String[] valores_recuperar = {"callID", "cSeq", "ipAddress", "name",
"port","method","message"};
        Cursor c = db.query("remensajes", valores_recuperar, seleccion, null, null, null, "cSeq
DESC", null);
        c.moveToFirst();
        do {
            Remensaje remensaje = new Remensaje(c.getString(0), c.getLong(1), c.getString(2),
c.getString(3), c.getString(4), c.getString(5), c.getString(6));
            lista_remensajes.add(remensaje);
        } while (c.moveToNext());
        db.close();
        c.close();
        return lista_remensajes;
    }


    public ArrayList<String> stringDecod (String stringCoded){

        String[] partes = stringCoded.split("::");
        ArrayList<String> stringDecod = new ArrayList<>(Arrays.asList(partes));

        return stringDecod;
    }

    public String stringCoded(ArrayList<String> stringDecod){

        StringJoiner joiner = new StringJoiner("::");
```

```
        for (String elemento : stringDecod) {
            joiner.add(elemento);
        }

        String stringCoded = joiner.toString();

        return stringCoded;
    }

    public void borrarTodo(){
        this.borrarRemensajes();
        this.borrarAutomaticas();
        this.borrarPlantillas();
    }


}
```

7.2.11 Automática

```
package com.example.sclock.basedatos;


public class Automatica {

    private String nombreAlarma;
    private String argumentosValor;
    private String tiempo;


    public Automatica() {
        //Constructor sin parámetros
    }
    public Automatica(String nombreAlarma, String argumentosValor, String tiempo) {
        this.nombreAlarma = nombreAlarma;
        this.argumentosValor = argumentosValor;
        this.tiempo = tiempo;
    }

    public String getNombreAlarma() {
        return nombreAlarma;
    }
    public void setNombreAlarma(String nombreAlarma) {
        this.nombreAlarma = nombreAlarma;
    }

    public String getArgumentosValor() {
        return argumentosValor;
    }
    public void setArgumentosValor(String argumentosValor) {this.argumentosValor =
argumentosValor;}

    public String getTiempo() {
        return tiempo;
    }
    public void setTiempo(String tiempo) {
        this.tiempo = tiempo;
    }
}
```

7.2.12 Plantilla

```
package com.example.sclock.basedatos;


public class Plantilla {

    private String nombreAlarma;
    private String autoManual;
    private String argumentos;
    private String argumentosSet;


    public Plantilla() {
```

```
        //Constructor sin parámetros
    }
    public Plantilla(String nombreAlarma, String autoManual, String argumentos, String
argumentosSet) {
        this.nombreAlarma = nombreAlarma;
        this.autoManual = autoManual;
        this.argumentos = argumentos;
        this.argumentosSet = argumentosSet;
    }

    public String getNombreAlarma() {
        return nombreAlarma;
    }
    public void setNombreAlarma(String nombreAlarma) {
        this.nombreAlarma = nombreAlarma;
    }

    public String getAutoManual() {
        return autoManual;
    }
    public void setAutoManual(String autoManual) {
        this.autoManual = autoManual;
    }

    public String getArgumentos() {
        return argumentos;
    }
    public void setArgumentos(String argumentos) {
        this.argumentos = argumentos;
    }

    public String getArgumentosSet() {
        return argumentosSet;
    }
    public void setArgumentosSet(String argumentosSet) {
        this.argumentosSet = argumentosSet;
    }
}
```

7.2.13 Remensaje

```
package com.example.sclock.basedatos;


public class Remensaje {

    private String callID;
    private Long cSeq; //orden tuplas
    private String ipAddress;
    private String name;
    private String port;
    private String method;
    private String message;


    public Remensaje() {
        //Constructor sin parámetros
    }
    public Remensaje(String callID, Long cSeq, String ipAddress, String name, String port, String
method, String message) {
        this.callID = callID;
        this.cSeq = cSeq;
        this.ipAddress = ipAddress;
        this.name = name;
        this.port = port;
        this.method = method;
        this.message = message;
    }

    public String getCallID() {
        return callID;
    }
    public void setCallID(String callID) {
        this.callID = callID;
    }
```

```
    public Long getCSeq() {
        return cSeq;
    }
    public void setCSeq(Long cSeq) {
        this.cSeq = cSeq;
    }

    public String getIpAddress() {
        return ipAddress;
    }
    public void setIpAddress(String ipAddress) {
        this.ipAddress = ipAddress;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public String getPort() {return port;}
    public void setPort(String port) {
        this.port = port;
    }

    public String getMethod(){return method;}
    public void setMethod(String method){this.method = method;}

    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }

}
```

7.2.14 xsd_scaip.xsd

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
    <xs:element name="mrq">
        <xs:complexType>
            <xs:all>
                <xs:element name="ref">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:pattern value="[0-9a-zA-Z]{1,16}"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="ver" minOccurs="0">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:pattern value="[0-9]{2}\.[0-9]{2}" />
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="sco" minOccurs="0">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:pattern value="[1-3]{1}" />
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="cha" minOccurs="0">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:pattern value="[0-9]{1}" />
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="mty" minOccurs="0">
```

```
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:pattern value="(ME)|(RE)|(IN)|(PI)" />
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="hbo" minOccurs="0">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:pattern value="[1-3]{1}" />
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="cid" minOccurs="0">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:pattern value="[0-9]{1,16}" />
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="dty" minOccurs="0">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:pattern value="[0-9]{1,4}" />
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="did" minOccurs="0">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:minLength value="1"/>
                <xs:maxLength value="8"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="dco" minOccurs="0">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:pattern value="[0_9]{1,3}" />
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="dte" minOccurs="0">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:minLength value="1"/>
                <xs:maxLength value="32"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="crd" minOccurs="0">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:minLength value="1"/>
                <xs:maxLength value="256"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="stc" minOccurs="0">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:pattern value="[0-9]{1,4}"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="stt" minOccurs="0">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:minLength value="1"/>
                <xs:maxLength value="32"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
    <xs:element name="pri" minOccurs="0">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:pattern value="[0-9]{1}"/>
            </xs:restriction>
```

```
                </xs:simpleType>
            </xs:element>
            <xs:element name="lco" minOccurs="0">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:pattern value="[0-9]{1,3}"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="lva" minOccurs="0">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                        <xs:maxLength value="2"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="lge" minOccurs="0">
                <xs:complexType mixed="true">
                    <xs:all>
                        <xs:element name="geo" minOccurs="0">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:pattern value="[0-9\.,]{1,23}"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                        <xs:element name="tim" minOccurs="0">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:pattern value="[0-9\-:\+T]{22}"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                        <xs:element name="gga" minOccurs="0">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:minLength value="0"/>
                                    <xs:maxLength value="81"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:element>
                    </xs:all>
                </xs:complexType>
            </xs:element>
            <xs:element name="lte" minOccurs="0">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                        <xs:maxLength value="32"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="ico" minOccurs="0">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:pattern value="[0-9]{1,3}"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="ite" minOccurs="0">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:minLength value="1"/>
                        <xs:maxLength value="128"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="ame" minOccurs="0">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:pattern value="[01]?"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
        </xs:all>
    </xs:complexType>
```

```
    </xs:element>
    <xs:element name="mrs">
        <xs:complexType>
            <xs:all>
                <xs:element name="ref">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:pattern value="[0-9a-zA-Z]{1,16}"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="snu">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:pattern value="[0-9]{1,5}"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="ste" minOccurs="0">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:minLength value="1"/>
                            <xs:maxLength value="128"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="cve" minOccurs="0">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:pattern value="[0-9]{2}\.[0-9]{2}"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="mre" minOccurs="0">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:pattern value="[0-9]{2}"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="cre" minOccurs="0">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:pattern value="[0-9]{2}"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="tnu" minOccurs="0">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:minLength value="1"/>
                            <xs:maxLength value="256"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
                <xs:element name="hbi" minOccurs="0">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:pattern value="[0-4]{1}"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
            </xs:all>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

7.2.15 activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.wear.widget.BoxInsetLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
    android:padding="@dimen/box_inset_layout_padding"
    tools:context=".Vistas.MainActivity"
    tools:deviceIds="wear">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="@dimen/inner_frame_layout_padding"
        app:layout_boxedEdges="all">

        <ImageView
            android:id="@+id/imageView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="GoMenu"
            android:src="@drawable/logosclock" />
    </FrameLayout>
</androidx.wear.widget.BoxInsetLayout>
```

7.2.16 lista_layout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.wear.widget.WearableRecyclerView
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/recyclerid">

</androidx.wear.widget.WearableRecyclerView>
```

7.2.17 lista_layout_button.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/buttonlista"
        android:layout_width="180dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="5dp"
        android:layout_marginBottom="5dp"
        android:text="Button"
        android:background="@drawable/round"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.596"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

7.2.18 simple_alarm_layout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.wear.widget.BoxInsetLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_boxedEdges="all">

        <TextView
            android:id="@+id/text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="35dp"
            android:text="Text Text" />
```

```
        <Button
            android:id="@+id/sendButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:layout_marginLeft="35dp"
            android:text="Send"
            />

        <Button
            android:id="@+id/backButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="100dp"
            android:layout_marginLeft="35dp"
            android:text="Back" />

    </FrameLayout>
</androidx.wear.widget.BoxInsetLayout>
```

7.2.19 choice_alarm_layout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.wear.widget.BoxInsetLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/box_inset_layout_padding"
    tools:context=".Vistas.MainActivity"
    tools:deviceIds="wear">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="@dimen/inner_frame_layout_padding"
        app:layout_boxedEdges="all">

        <TextView
            android:id="@+id/text"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="0dp"
            android:text="Text Text" />

        <Button
            android:id="@+id/firstButton"
            android:layout_width="50dp"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dp"
            android:background="@drawable/round"
            android:text="1" />

        <Button
            android:id="@+id/secondButton"
            android:layout_width="50dp"
            android:layout_height="wrap_content"
            android:layout_marginLeft="50dp"
            android:layout_marginTop="20dp"
            android:background="@drawable/round"
            android:text="2" />
        <Button
            android:id="@+id/thirdButton"
            android:layout_width="50dp"
            android:layout_height="wrap_content"
            android:layout_marginLeft="100dp"
            android:layout_marginTop="20dp"
            android:background="@drawable/round"
            android:text="5" />
        <Button
            android:id="@+id/fourthButton"
            android:layout_width="50dp"
            android:layout_height="wrap_content"
            android:layout_marginLeft="25dp"
            android:layout_marginTop="70dp"
            android:background="@drawable/round"
```

```
            android:text="10" />
        <Button
            android:id="@+id/fifthButton"
            android:layout_width="50dp"
            android:layout_height="wrap_content"
            android:layout_marginLeft="75dp"
            android:layout_marginTop="70dp"
            android:background="@drawable/round"
            android:text="0" />
        <Button
            android:id="@+id/backButton"
            android:layout_width="wrap_content"
            android:layout_height="40dp"
            android:layout_marginLeft="30dp"
            android:layout_marginTop="120dp"
            android:text="Back" />



    </FrameLayout>
</androidx.wear.widget.BoxInsetLayout>
```

7.2.20 round.xml

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">
    <solid android:color="#03ae3c"/>
</shape>
```

7.2.21 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.sclock">

    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

    <uses-feature android:name="android.hardware.type.watch" />

    <application
        android:name=".SCAIP.SCAIPWebApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@android:style/Theme.DeviceDefault">
        <uses-library
            android:name="com.google.android.wearable"
            android:required="true" />

        <!--
               Set to true if your app is Standalone, that is, it does not require the handheld
               app to run.
        -->
        <meta-data
            android:name="com.google.android.wearable.standalone"
            android:value="true" />

        <activity
            android:name=".Vistas.MainActivity"
            android:exported="true"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

```
        <activity
            android:name=".Vistas.ListaBotones">
        </activity>
        <activity
            android:name=".Vistas.SimpleAlarmView">
        </activity>
        <activity
            android:name=".Vistas.ChoiceAlarmView">
        </activity>

    </application>

</manifest>
```

7.2.22 build.gradle

```
plugins {
    id 'com.android.application'
}

android {
    compileSdk 32

    defaultConfig {
        applicationId "com.example.sclock"
        minSdk 25
        targetSdk 32
        versionCode 1
        versionName "1.0"
    }


    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-
rules.pro'
        }
    }
    buildFeatures {
        viewBinding true
    }
}

dependencies {

    implementation group: 'javax.sip', name: 'android-jain-sip-ri', version: '1.3.0-91'
    implementation group: 'log4j', name: 'log4j', version: '1.2.17'
    implementation group: 'de.mindpipe.android', name: 'android-logging-log4j', version: '1.0.3'
    implementation 'com.google.android.gms:play-services-wearable:17.1.0'
    implementation 'androidx.percentlayout:percentlayout:1.0.0'
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
    implementation 'androidx.recyclerview:recyclerview:1.2.1'
    implementation 'androidx.wear:wear:1.1.0'
    implementation 'xerces:xercesImpl:2.12.0'
    implementation 'org.bouncycastle:bcprov-jdk15on:1.69'
    implementation 'commons-codec:commons-codec:1.15'

}
```