Graduado en Ingeniería Informática

Universidad a Distancia de Madrid

Departamento de Ingeniería Informática

TRABAJO DE FIN DE GRADO

Sistema de alarma monitorizado y telemandado

Autor: Fidel Brea Montilla

Director: David Lizcano Casas

MADRID, JULIO DE 2022

Declaración de originalidad:

El autor de este trabajo, Fidel Brea Montilla, con D.N.I. 50.976.906-M, declara que el contenido de este trabajo de fin de grado es original, y en el caso de haber utilizado las ideas o contenidos de otros trabajos de otros autores están convenientemente citados, de acuerdo con las normas de citación establecidas.

El número de palabras de este trabajo, excluyendo los anexos es: 13.683

Índice

Índice de figuras

Índice de tablas

Resumen: (máximo 1 página)

Palabras clave: (máximo 5 palabras, separadas por comas)

Abstract:

Keywords:

CAPÍTULO 1: INTRODUCCIÓN

1.1. Planteamiento del problema

Según refleja el Ministerio del Interior en su último anuario estadístico (Ministerio del Interior, 2021), los delitos contra el patrimonio es la categoría que acumula la mayoría de los delitos del conjunto de criminalidad conocidas por las Fuerzas y Cuerpos de Seguridad (FCS) situando la incidencia de este tipo de delitos alrededor del 75% del total de los delitos cometidos. De los cuales el 8,66% son robos en viviendas y establecimientos.

Tabla 1: Cantidad de robos en viviendas y establecimientos por comunidades y ciudades autónomas en 2020. Fuente: Portal estadístico de criminalidad del Ministerio del Interior. Elaboración propia.

Comunidad autónoma	Cantidad	Proporción
Cataluña	26403	23,38 %
Andalucía	18884	16,72 %
Comunitat valenciana	15982	14,15 %
Madrid (comunidad de)	13961	12,36 %
País vasco	5385	4,77 %
Murcia (región de)	4954	4,39 %
Castilla - la mancha	4926	4,36 %
Galicia	4300	3,81 %
Canarias	3850	3,41 %
Castilla y león	3461	3,06 %
Balears (illes)	2790	2,47 %
Aragón	2025	1,79 %
Extremadura	1636	1,45 %
Cantabria	1414	1,25 %
Asturias (principado de)	1228	1,09 %
Navarra (comunidad foral de)	1103	0,98 %
Rioja (la)	475	0,42 %
Ciudad autónoma de melilla	84	0,07 %
Ciudad autónoma de ceuta	81	0,07 %

La Tabla 1 refleja el número de robos en viviendas y establecimientos por comunidades y ciudades autónomas en el año 2020, que es el periodo más reciente con datos consolidados. En esta tabla se puede ver cómo Cataluña, Andalucía, la Comunidad Valenciana y la

Comunidad Autónoma de Madrid (CAM) son las comunidades autónomas que registran un mayor número de este tipo de delitos representando las dos terceras partes del total de robos en viviendas y establecimientos de toda España. El resto de comunidades autónomas tienen una incidencia considerablemente inferior.

En la CAM se cometieron un total de 13.961 robos a lo largo del año 2020, por otro lado, según datos del Instituto de Estadística de la CAM, en el año 2020 había 3.029.616 viviendas en la comunidad (Instituto de estadística de la CAM, 2020), si cruzamos ambos datos podemos afirmar que en la CAM, durante el año 2020, una de cada 217 viviendas fue víctima de robo. La afirmación anterior no tiene en cuenta la repetitividad de los robos en una misma vivienda puesto que como se deriva del trabajo de investigación de Agustina & Reales (2013) con presos por delito de robo en vivienda, robar dos veces en la misma vivienda no es una práctica muy frecuente.

Del estudio realizado por Caballero et al. (2000) sobre los efectos que tiene este tipo de delito sobre los propietarios de las viviendas se desprende que una persona que es víctima de un delito de robo en su casa sufre una serie de reacciones psicológicas negativas que permanecen por más de un año y suponen para la víctima secuelas desagradables y dolorosas. De estas experiencias negativas, ya sean propias o ajenas, surge la necesidad de evitar el delito.

Entre las medidas más populares para evitar un robo en el domicilio destacan la instalación de mecanismos de bloqueo en los accesos (puertas blindadas o acorazadas, cerrojos, cerraduras de última generación, enrejado de puertas y ventanas, etcétera) y la instalación de un sistema de alarma. Si bien la mayoría de ladrones afirma que la presencia de un sistema de alarma en una vivienda no les disuade de cometer el delito, sí afirman reducir drásticamente el tiempo de actuación cuando se ven sorprendidos por el disparo de una alarma. Como referencia, un robo en una vivienda sin sistema de alarma (o en la que el sistema de alarma no se ha disparado) tiene una duración de actuación que oscila entre 10 y 15 minutos, mientras que si salta la alarma, la duración del robo se reduce a un intervalo de entre 3 y 5 minutos (Agustina, J. R. & Reales, F., 2013), esto pone de manifiesto que un

sistema de alarma se puede considerar como un elemento de seguridad eficaz puesto que tiene una incidencia directa en la reducción del tiempo de actuación del robo.

Securitas Direct España, Movistar Prosegur Alarmas, Tyco Integrated Security (alarmas ADT) y el Grupo INV Sistemas y Soluciones de Seguridad son las empresas que dominan el mercado de sistemas de alarma en España para hogares. Los sistemas de alarma que comercializan estas empresas utilizan el aire como medio compartido para la transmisión de información entre los distintos elementos. El uso de radiofrecuencias para interconectar las distintas partes del sistema de alarma presenta la gran ventaja para el cliente de no requerir ningún tipo de obra para su instalación, pero también presenta dos grandes inconvenientes, por un lado, no es inmune a los inhibidores de frecuencias, y por otro lado, depende de pilas para el funcionamiento de los elementos periféricos

Los sistemas de alarma más vendidos se comercializan con un servicio que implica el abono de una cuota. Dicho servicio consiste en la conexión de la alarma a un centro de control que verificará el disparo de la alarma y avisará a las FCS en caso de robo. Sin embargo, estas empresas están supeditadas a la Ley 5/2014, de 4 de abril, de Seguridad Privada (que deroga la Ley 23/1992, de 30 de julio) y el funcionamiento de sus sistemas de alarma están regulados por la Orden INT/316/2011, de 1 de febrero, sobre funcionamiento de los sistemas de alarma en el ámbito de la seguridad privada. Esta orden especifica cuatro mecanismos de verificación de alarmas para considerar a ésta como válida y poder avisar y movilizar a las FCS (Orden INT/316/2011. Capítulo II. Comunicación de alarmas):

Verificación secuencial, consistente en la activación de manera secuencial coherente de al menos tres sensores diferentes en un intervalo de tiempo determinado.

Verificación mediante vídeo, consistente en el envío de una imagen del momento del disparo de la alarma y al menos otras dos posteriores capturadas dentro de los cinco segundo siguientes al disparo.

Verificación mediante audio, consistente en enviar una grabación de audio de al menos 20 segundos antes del disparo.

Verificación personal, consistente en una comprobación humana por parte de un vigilante de seguridad uniformado que acudirá en vehículo con anagrama de la empresa de seguridad.

El mecanismo de actuación de los centros de control antes de contactar con los FCS es lento y muchas veces ineficaz ya que se centran en verificaciones secuenciales que nunca se dan porque no salta más de uno o dos sensores, e imágenes de vídeo que nunca llegan debido al uso de inhibidores de señales por parte de los delincuentes. De este modo se puede considerar como placebo el servicio proporcionado por estas empresas de alarma.

1.2. Objetivos

El objetivo de este trabajo es desarrollar un sistema de alarma para una vivienda que, además del aviso acústico local, avise al propietario empleando tecnologías actuales. El sistema de alarma deberá ser inmune a los inhibidores de frecuencias y el propietario (usuario) del sistema deberá contar con una herramienta que le permita monitorizar y manipular el sistema de alarma desde cualquier lugar.

El desarrollo del sistema se llevará a cabo empleando técnicas y tecnologías vistas en diversas asignaturas de la carrera con el fin de aplicarlas de manera práctica y también se emplearán tecnologías y herramientas punteras.

CAPÍTULO 2: ESTADO DEL ARTE

2.1. Marco legal actual de los sistemas de alarma

El concepto de seguridad privada hace alusión a las empresas que prestan servicios de seguridad y no es aplicable a una persona que instala su propio sistema de alarma en una vivienda, sin embargo, en el ámbito de la seguridad privada encontramos un marco legal que proporciona especificaciones y protocolos que ayudarán a establecer requisitos a la hora de confeccionar un sistema de alarma.

Actualmente, el funcionamiento de los sistemas de alarma en el ámbito de la seguridad privada está legislado por la Orden INT/316/2011, de 1 de febrero, sobre funcionamiento de los sistemas de alarma en el ámbito de la seguridad privada. Esta Orden Ministerial regula entre otras cosas las características, especificaciones y diversos protocolos que deben cumplir los sistemas de alarma y las empresas de seguridad privada para integrarse así con las diferentes normas europeas. En su artículo 2 define los cuatro grados de seguridad en los que se clasifican los sistemas de alarma según recoge la Norma UNE-EN 50131-1.

Grado 1. Sistemas de alarma con señalización acústica sin conexión a una central de alarmas o a un centro de control.

Grado 2. Viviendas y pequeños establecimientos, comercios e industrias en general, conectados a una central de alarmas o a un centro de control.

Grado 3. Establecimientos obligados a disponer de medidas de seguridad y conectados a central de alarmas o a un centro de control.

Grado 4. Infraestructuras críticas.

Según esta clasificación, el sistema de alarma que pretende desarrollar este trabajo estará catalogado como Grado 1, sin embargo, para el diseño se considerará el Grado 2 puesto

que los elementos en el Grado 2 se fabrican con unos requisitos de seguridad mayores y en todo momento el usuario hará las veces de central de control. Es interesante que el usuario del sistema de alarma sea conocedor del contenido de esta Orden Ministerial.

2.2. Detectores de apertura de puertas y ventanas

Figura 1: Detector de apertura. Fuente: Naylamp Mechatronics

Los detectores de apertura de puertas y ventanas se componen de dos partes, una de ellas móvil y la otra fija. La parte fija tiene el cableado y contiene en su interior un *reed* o interruptor magnético, la parte móvil contiene un imán que acciona el *reed*. Cuando ambas partes se encuentran enfrentadas, el *reed* cierra un contacto eléctrico libre de potencial que se utiliza para indicar un estado de normalidad a la central de alarma. Dicho contacto se abre para indicar que la normalidad se ha roto.

2.3. Detectores de movimiento

Para detectar movimiento dentro de un espacio determinado existen principalmente dos estrategias: detección activa y detección pasiva. La estrategia de detección activa consiste en inyectar algún tipo de señal en el espacio a vigilar e interpretar el eco producido teniendo en cuenta el efecto doppler. La inyección de ultrasonidos es la técnica más empleada para implementar esta estrategia en los detectores de interior (Hidalgo Garcés, R., 1980). Por otro lado, la estrategia de detección pasiva consiste en la interpretación de algún tipo de señal proveniente de los cuerpos que se encuentran en el espacio vigilado sin inyectar ningún tipo de señal al medio. Las dos técnicas utilizadas en los sistemas de alarma para implementar esta estrategia son, por un lado, los sensores Passive Infra Red (PIR) que captan las ondas infrarrojas emitidas por los cuerpos (Fuertes Araque, J., 2007); y por otro lado, la detección de movimiento en imágenes captadas por una cámara.

Los detectores de movimiento proporcionan un contacto eléctrico normalmente cerrado libre de potencial. Este contacto permanecerá cerrado mientras no detecte movimiento y será abierto durante unos pocos segundos cuando detecte movimiento.

2.4. Tampers

Un tamper es un sistema de protección antisabotaje. La protección antisabotaje consiste en controlar el acceso al interior del elemento y por lo tanto a los sistemas que lo accionan. Esta protección antisabotaje se implementa mediante un contacto eléctrico que permanece en estado cerrado mientras la protección del elemento es correcta. Dicho contacto eléctrico recibe el nombre de tamper y se pueden cablear en serie con el contacto normalmente cerrado del elemento que protege o se pueden enviar de forma individual a la central de alarma.

En los sistemas de grado 2 es obligatorio su uso en paneles, equipos de señalización, fuentes de alimentación, detectores y cajas de paso (Orden INT/316/2011, de 1 de febrero, sobre funcionamiento de los sistemas de alarma en el ámbito de la seguridad privada).

2.5. Sirenas de interior

El Código Penal español recogido en la Ley Orgánica 10/1995, de 23 de noviembre, del Código Penal y modificado en dos ocasiones, una por la Ley Orgánica 15/2003, de 25 de noviembre, por la que se modifica la Ley Orgánica 10/1995, de 23 de noviembre, del Código Penal, y otra por la Ley Orgánica 1/2015, de 30 de marzo, por la que se modifica la Ley Orgánica 10/1995, de 23 de noviembre, del Código Penal, dedica el título XIII a definir los delitos y penas contra el patrimonio y contra el orden socioeconómico. De los artículos 237 y 238 se concluye que toda persona que roba en una vivienda o establecimiento es reo del delito de robo, y de los artículos 241 y 242 se deriva una pena mínima de dos años, por lo que no sería posible sustituir dicha pena por multa o multa y trabajos en beneficio de la comunidad, asegurando así el ingreso en prisión. La magnitud de la pena recogida por el Código Penal para el delito de robo en vivienda es importante puesto que supone una medida disuasoria.

Las sirenas en un sistema de alarma tienen un objetivo claro: dar la voz de alarma. Esto no le pasa desapercibido al ladrón que, tal y como recoge el estudio realizado por Agustina & Reales (2013) sobre autores de robo en vivienda, reducirá el tiempo de actuación.

Las sirenas interiores comercializadas producen un sonido con una presión entre 95 y 110 dB a un metro de distancia, medida que está por debajo de los 140 dB que definen el umbral del dolor (Álvarez Bayona, T., 2008) y el cual puede producir lesiones irreversibles tales como recoge Ricketts (2022) en su artículo “Deafening Siren” (traducido, sirena ensordecedora) que menciona la pérdida auditiva neurosensorial en un oído tras una exposición fortuita no continuada a una fuente de sonido de 140 dB. Si el sistema de alarma causara algún tipo de daño al ladrón se estaría incurriendo en un delito de lesiones.

2.6. Sistema de alimentación ininterrumpido

Un Sistema de Alimentación Ininterrumpido (SAI), también conocido por su denominación inglesa Uninterruptible Power Supply (UPS), es un sistema encargado de proporcionar energía eléctrica a otro sistema aun cuando la fuente principal de suministro de energía falle. La fuente de energía secundaria proviene de una o más baterías (Choque Zapana, M. A., 2018).

La alimentación principal de los sistemas de alarma es típicamente de 12 voltios nominales en corriente continua (12V DC) que hará funcionar a los distintos elementos. La implementación de un SAI para una tensión de 12V DC es sencilla frente a otras tensiones debido a las características de las propias baterías.

Según las especificaciones de la Orden INT/316/2011, de 1 de febrero, sobre funcionamiento de los sistemas de alarma en el ámbito de la seguridad privada, un sistema de alarma de Grado 2 debe proporcionar una autonomía de al menos 12 horas tras un corte de suministro de energía eléctrica primaria, además, debe ser capaz de recargar en un plazo máximo de 72 horas el 80% de la carga necesaria para proporcionar esas 12 horas de autonomía.

2.7. Etiquetas RFID

Las etiquetas Radio Frequency Identification (RFID) se utilizan para identificar objetos de diversa índole cuyo principio de funcionamiento consiste en introducir la antena de la

etiqueta RFID dentro de un campo electromagnético, esto energiza un pequeño circuito integrado contenido en la etiqueta RFID que transmite su información (Alexandres, S. et al, 2006). Este tipo de dispositivos se emplea en los sistemas de alarma para efectuar el desarme del sistema. Cada usuario suele poseer una etiqueta RFID en forma de llavero, de tarjeta o pegatina, de este modo, como cada etiqueta RFID tiene un número de serie unívoco, se puede identificar al usuario que realizó la acción. Desde el punto de vista de la seguridad informática, la robustez de este sistema se basa en el tiempo que se tardaría en dar con los números de serie de los llaveros aplicando la fuerza bruta.

2.8. ONVIF

La Open Network Video Interface Forum (ONVIF) es una organización que proporciona y promueve interfaces estandarizadas para operar con productos de seguridad basados en IP. ONVIF define una serie de perfiles y cada perfil tiene un conjunto de características particulares que se aplicarán a un producto concreto. Los perfiles definidos actualmente por ONVIF son:

Perfil A, configuración de control de acceso

Perfil C, control de puertas y gestión de eventos

Perfil D, periféricos de control de acceso

Perfil G, almacenamiento y recuperación en el borde

Perfil M, metadatos y eventos para aplicaciones de análisis

Perfil Q, instalación rápida

Perfil S, transmisión de vídeo básica

Perfil T, transmisión de video avanzada

Cada perfil tiene una serie de funciones que deben ser implementadas por el dispositivo, unas tienen carácter obligatorio y otras son opcionales. Conociendo las funciones del perfil del producto con el que se trabaja se puede interactuar con él. La comunicación con el dispositivo se realiza utilizando el protocolo de acceso a objetos Simple Object Access

Protocol (SOAP), el cual usa el lenguaje Extensible Markup Language (XML) y el método POST del protocolo Hypertext Transfer Protocol (HTTP) para el intercambio de mensajes.

Por ejemplo, si se quisiera conseguir la URI (Uniform Resource Identifier) para acceder a la imagen más actual de un dispositivo en formato JPEG, dicho dispositivo tendría que implementar el perfil T (ONVIF, 2018), el cual contempla la función GetSnapshotUri. El mensaje de petición que enviaría el cliente y su mensaje de respuesta tendrían un aspecto similar al siguiente:

Mensaje de petición enviado al dispositivo:

```
<env:Envelope <?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope [...]>
<env:Header>
  <wsse:Security>
  <wsse:UsernameToken>
    <wsse:Username>admin</wsse:Username>
    <wsse:Password Type="http[...]">dax4B6116bx8JHp3ExdAwuuWlx8=</wsse:Password>
    <wsse:Nonce EncodingType="http[...]Base64Binary">LTEzMjIwNDcwNzc=</wsse:Nonce>
    <wsu:Created>2022-03-14T17:37:27Z</wsu:Created>
  </wsse:UsernameToken>
</wsse:Security>
</env:Header>
<env:Body>
  <GetSnapshotUri xmlns="http://www.onvif.org/ver10/media/wsdl">
  <ProfileToken>MainProfileToken</ProfileToken>
</GetSnapshotUri>
</env:Body>
</env:Envelope>
```

Texto 1: Mensaje de petición de URI

Mensaje de respuesta del dispositivo:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope [...]>
<SOAP-ENV:Header>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
    <trt:GetSnapshotUriResponse>
       <trt:MediaUri>
          <tt:Uri>http://[IP]:8080/web/auto.jpg?-usr=admin&amp;-pwd=admin</tt:Uri>
          <tt:InvalidAfterConnect>false</tt:InvalidAfterConnect>
          <tt:InvalidAfterReboot>false</tt:InvalidAfterReboot>
          <tt:Timeout>PT0H12M0S</tt:Timeout>
      </trt:MediaUri>
   </trt:GetSnapshotUriResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Texto 2: Mensaje de respuesta de URI

La URI obtenida es `http://192.168.1.38:8080/web/auto.jpg?-usr=admin&-pwd=admin`

2.9. Streaming de vídeo mediante RTSP

La mayoría de cámaras con conexión a redes TCP/IP (cámaras IP) ofrecen la transmisión de flujo de datos (audio y vídeo) empleando el protocolo Real Time Streaming Protocol (RTSP) el cual se encuentra actualmente definido en su versión 2.0 por el documento RFC7826 (Schulzrinne et al., 2016). El protocolo RTSP define un intercambio de mensajes entre cliente y servidor a nivel de aplicación con una sintaxis similar al definido por HTTP, puede funcionar sobre los protocolos de transporte TCP, UDP y RDP aunque la mayoría de dispositivos lo implementan sobre TCP. Este protocolo se encarga básicamente de negociar cómo se va a realizar la transmisión con los comandos DESCRIBE y SETUP y, una vez iniciada, se encarga de controlar la reproducción mediante los mensajes PLAY, PAUSE y TEARDOWN. Normalmente se usa los protocolos RTP sobre UDP y RTCP para la transmisión del contenido tal y como indican Mateos Costilla y Reaño Montoro (2008).

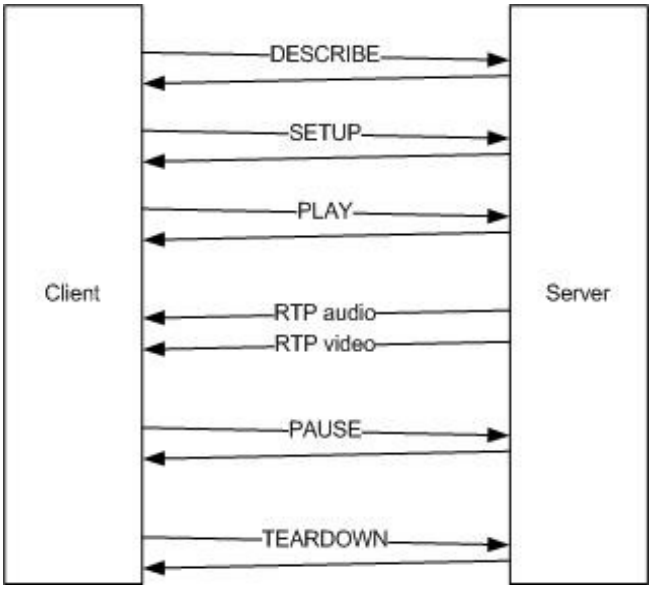


Figura 2: Diagrama de secuencia de ejemplo de control y streaming.
Fuente: W3C Media Fragments Working Group

La Figura 2 muestra un diagrama de secuencia de una conexión típica a un servidor RTSP en la que se observa cómo inicialmente se negocia el recurso y forma de transmisión (mensajes DESCRIBE y SETUP) y posteriormente se inicia la reproducción (PLAY) y, tras reproducir el contenido, se detiene y finaliza la conexión (PAUSE y TEARDOWN).

2.10. Protocolo Wiegand

John Richard Wiegand bautizó con su nombre al fenómeno observado en el comportamiento en ciertas partes de un hilo conductor influido por un campo magnético, este fenómeno se conoce como **efecto Wiegand** y ha sido utilizado entre otras cosas para crear un tipo de tarjetas de identificación que contienen en su interior pequeños trozos de hilos conductores dispuestos en dos filas de tal forma que producen una sucesión de bits al deslizarse a través de un campo electromagnético generado por dos bobinas (Álvarez Villavicencio, C. J., 1991), una de estas filas representa los bits 0 que serán captados por una bobina, y la otra fila representa los bits 1 que serán captados por la otra bobina tal y como se puede apreciar en la Figura 3, la cual representa una tarjeta de identificación con el código 010100001110100.

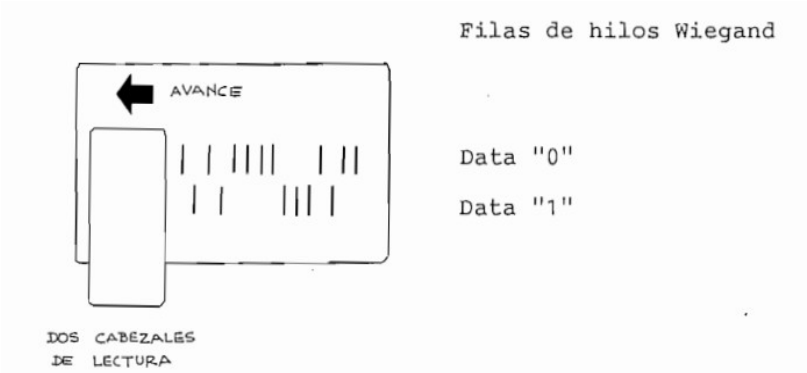


Figura 3: Representación de tarjeta de identificación de 15 bits atravesando el módulo con las dos cabezas de lectura. Fuente: Álvarez, C. J. (1991)

El **protocolo Wiegand** nace para dar solución a la transmisión de la información captada por las dos bobinas del lector de tarjetas. Es un protocolo unidireccional en el que un emisor emite una información que recibe el receptor y la interpreta. Actualmente, el protocolo Wiegand es el método predominante mediante el cual los controles de acceso envían la información al dispositivo conectado.

Figura 4: Control de acceso modelo Wiegand 34 bits KR602-M con lectura de etiquetas RFID 13,56 MHz. Fuente: Jiji.ng

Las especificaciones eléctricas y lógicas del protocolo de comunicación Wiegand están reflejadas en el estándar AC-01 de la Security Industry Association (SIA), respaldado por el American National Standards Institute (ANSI) y titulado “Protocolo estándar de control de acceso para la interfaz de lector Wiegand de 26 bits” (Security Industry Association [SIA], 1996). Esta especificación de la SIA ha sido utilizada para transmitir mensajes de longitudes distintas a los 26 bits. Actualmente, los formatos de 4 y 8 bits se emplean para transmitir pulsaciones de teclados numéricos y los formatos de 26 y 34 bits para transmitir las identificaciones de las lecturas de etiquetas RFID.

El estándar Wiegand de la SIA especifica el uso de cinco cables. Dos de ellos se emplean para alimentar al control de acceso, otros dos para transmitir datos (Data0 y Data1) y uno para señalización (LEDCTL). Este último suele activar una señalización luminosa para proporcionar retroalimentación al usuario.

Los cables para transmitir los datos (Data0 y Data1) son salidas de colector abierto, lo que significa que se adapta a la tensión de trabajo del dispositivo al que se conecta, éste deberá contar con una resistencia pull-up en cada salida para su funcionamiento tal y como muestra el esquema de la Figura 5, de este modo la señal V_{OUT} tendrá un nivel bajo (cero voltios) si el transistor se encuentra en saturación y

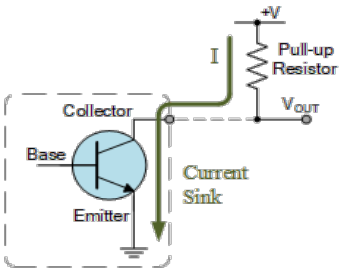


Figura 5: Salida colector abierto con resistencia pull-up. Fuente: Electronics Tutorials

tendrá un nivel alto (+V) si el transistor se encuentra en corte. Tanto Data0 y Data1 son señales activas a nivel bajo, esto significa que permanecen constantemente a un nivel eléctrico alto (+V) y basculan a cero voltios cuando desean enviar el bit correspondiente.

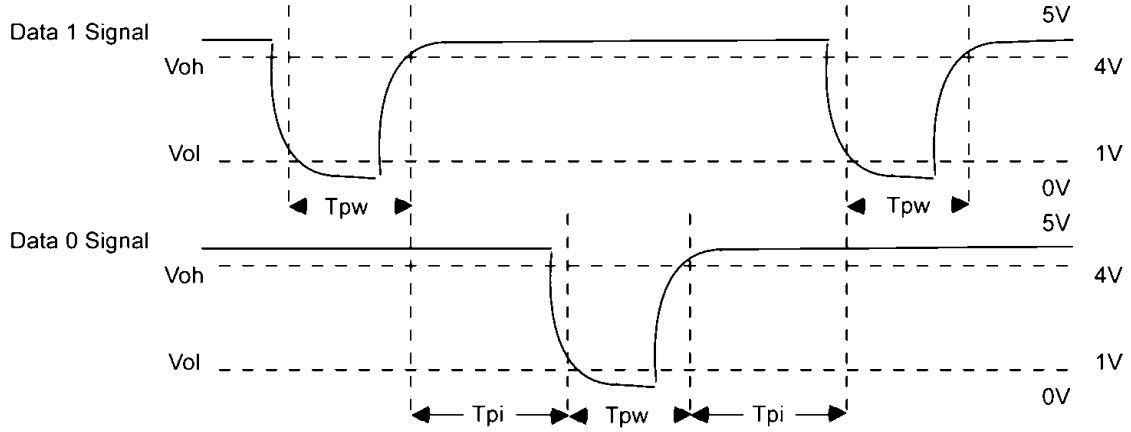


Figura 6: Detalle de pulsos de las señales Data0 y Data1 y su duración.
Fuente: Security Industry Association (1996).

La Figura 6 ilustra la transmisión del código 101. En esta figura se observa cómo para definir un pulso en cualquiera de las líneas (Data0 y Data1), la tensión debe bajar primero del umbral de 1 voltio y después subir por encima del umbral de 4 voltios. Según refleja la especificación de la Security Industry Association (1996), la duración del ancho de pulso (Tpw) puede oscilar entre los 20 y 100 microsegundos y la separación entre pulsos (Tpi) puede oscilar entre los 200 microsegundos y los 20 milisegundos, si bien los valores típicos serán de 50 microsegundos y 1 milisegundo respectivamente.

2.10.1 Wiegand 4 bits y 8 bits para teclados

Los teclados de los controles de acceso suelen utilizar el formato de 4 bits, en el que transmiten la tecla pulsada empleando un nibble (conjunto de 4 bits) o el formato de 8 bits, el cual emplea la misma codificación que el formato de 4 bits pero añade un segundo nibble, en la parte más significativa, que se corresponde con el primer nibble invertido tal y como muestra la Tabla 2. De este modo, el formato de 8 bits proporciona un mecanismo de integridad embebido en el propio código (Farpointe Data, Inc. 2020).

Tabla 2: Códigos Wiegand de 4 y 8 bits para teclados. Elaboración propia.

Tecla	Código 4 bits	Código 8 bits
0	0000	11110000
1	0001	11100001
2	0010	11010010
3	0011	11000011
4	0100	10110100
5	0101	10100101
6	0110	10010110
7	0111	10000111
8	1000	01111000
9	1001	01101001
*	1010	01011010
#	1011	01001011

2.10.2 Wiegand 26 bits

Si bien hay libertad en definir un formato propio, el formato de 26 bits especificado por la SIA es el siguiente (SIA, 1996) y (Farpointe Data, Inc. 2019).

B0, paridad par del conjunto de bits B1-B12.

B1-B8, facility code (código de fabricante), donde B1 es el bit más significativo.

B9-B24, user code (código de usuario), donde B9 es el bit más significativo.

B25, paridad impar del conjunto de bits B13-B24.

2.10.3 Wiegand 34 bits

No existe ninguna definición formal para este formato. La mayoría de controles de acceso extienden la especificación existente para el formato de 26 bits empleando el primer y último bit para paridad y los 32 bits centrales para el código de la etiqueta RFID.

2.11.Arduino Mega 2560

Figura 7: Chip ATMEGA2560. Fuente: Atmel

La placa Arduino MEGA está basada en el microcontrolador ATmega2560 de Atmel (propiedad de Microchip Technology). El ATmega2560 es un microcontrolador RISC perteneciente a la familia de microcontroladores AVR (Advanced Virtual RISC) de Atmel diseñada por Alf-Egil Bogen y Vegard Wollan en los años 90. Tal y como se puede leer en la nota técnica elaborada por Bogen & Wollan (1999) y la hoja de

datos del ATmega2560 (Microchip Technology, 2022), los microcontroladores AVR de Atmel tienen un bus de datos de 8 bits, 32 registros de propósito general, un diseño de arquitectura orientada a la optimización de programas elaborados en lenguaje C y son capaces de ejecutar cada instrucción en un ciclo de reloj lo que proporciona rendimientos de un millón de instrucciones por segundo (MIPS) por cada megahercio, todo ello con un consumo energético muy reducido y proporcional a la velocidad de procesamiento (ratio 1:1) gracias a la tecnología CMOS.

El microcontrolador ATmega2560 cuenta con una memoria flash de 256 KiB de los cuales 8 KiB están ocupados por el cargador de arranque (bootloader) que usa Arduino para cargar los programas compilados usando el protocolo serie STK500 propio de Atmel.

También se pone a disposición del usuario una memoria EEPROM de 4 KiB para almacenamiento de datos no volátiles.

La memoria RAM del ATmega2560 es de 8 KiB, y la tecnología empleada es SRAM (Static RAM), que presenta una mayor velocidad y menor consumo de energía frente a la DRAM (Dynamic RAM).

Electrónicamente pone a disposición del usuario un total de 70 pines de los cuales 54 son configurables individualmente como entradas o salidas con una capacidad de suministro de corriente eléctrica por salida de hasta 20 mA y los 16 pines restantes son entradas analógicas con una resolución de 10 bits. Parte del conjunto de los 54 pines configurables se pueden configurar para trabajar con un puerto Serial Peripheral Interface (SPI), un puerto Inter-Integrated Circuit (I^2C) y hasta cuatro puertos Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART).

La placa Arduino Mega 2560 Rev 3 hace funcionar al ATmega2560 a 16 MHz, que es la velocidad máxima indicada por el fabricante del microcontrolador, lo que redunda en un rendimiento de 16 MIPS. Esta placa está diseñada para facilitar el acceso al microcontrolador tanto

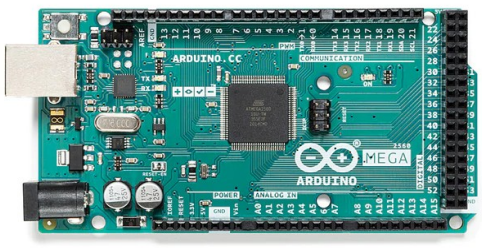
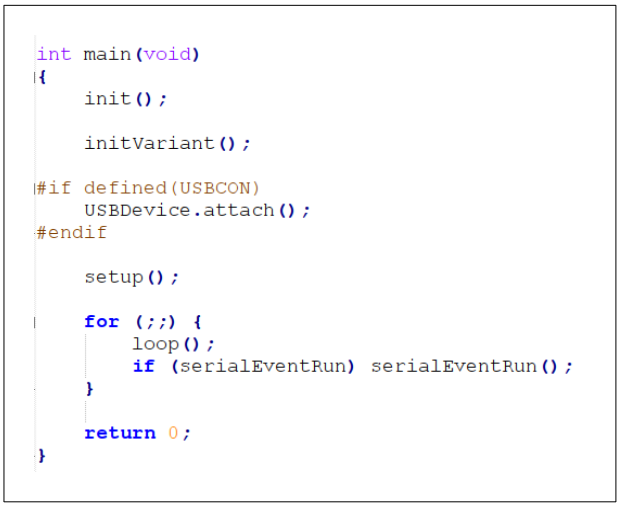


Figura 8: Placa Arduino Mega 2560 Rev 3. Fuente: arduino.cc

desde el punto de vista de su programación como el de la interconexión de sus distintos pines. Arduino dispone para su programación de un entorno de desarrollo (IDE). El IDE Arduino está desarrollado para trabajar con diferentes tarjetas y diferentes microcontroladores, por tanto, cuando el usuario compila su código, éste pasa a formar parte de un proyecto mayor que contempla las diferentes configuraciones de la tarjeta elegida.



```
int main(void)
{
    init();

    initVariant();

#if defined(USBCON)
    USBDevice.attach();
#endif

    setup();

    for (;;) {
        loop();
        if (serialEventRun) serialEventRun();
    }

    return 0;
}
```

Figura 9: Contenido del archivo main.cpp compilado por la IDE Arduino. Fuente: Biblioteca arduino

La función *main* del proyecto compilado se muestra en la Figura 9 y se puede observar que llama primero a la función *setup()* para después llamar un número interminable de veces a la función *loop()*. Ambas funciones deberán ser desarrolladas por el usuario en su código. También llama la atención el mecanismo de polling implementado para consultar la presencia de datos en los buffers de las UART a través de la llamada a la función *serialEventRun()* en cada vuelta.

2.12. Raspberry Pi 3

Las placas Raspberry Pi son computadoras embebidas en una única placa. Basan su funcionamiento en los circuitos integrados SoC (System on a Chip) del fabricante Broadcom. Los SoC integran todo o gran parte de un computador en su interior.

Se puede encontrar numerosas referencias a las placas Raspberry Pi en diferentes textos académicos debido a su relación rendimiento-coste para implementar distintos proyectos.

Para este trabajo se tendrá en cuenta la placa Raspberry Pi 3 por ofrecer un consumo energético mucho más reducido que el modelo superior, la Raspberry Pi 4. Además, la Raspberry Pi 3 ofrece un procesador de cuatro núcleos Cortex-A53 diseñado para trabajar con el juego de instrucciones RISC de 64 bits ARMv8 (Advanced RISC Machine). Para tener una referencia comparativa sobre el rendimiento de un sistema informático debemos acudir a los distintos bancos de prueba. Gareth Halfacree realiza un gran análisis a este respecto en dos artículos (2018, 2019) de los cuales podemos concluir que el modelo Raspberry Pi 3 B+ mejora algo en rendimiento y térmicamente a su predecesor, el modelo Raspberry Pi 3 B. Sin embargo, el consumo energético tanto en estado de inactividad como de actividad es notablemente mayor en el modelo B+. La placa más avanzada, la Raspberry Pi 4 B, presenta un rendimiento muy superior al modelo más avanzado de la Raspberry Pi 3, sin embargo, lo hace a costa de un consumo energético mayor y genera mucho más calor. Es conocido el problema térmico que presenta la placa Raspberry Pi 4, dicho problema ha sido mitigado con distintas actualizaciones (Halfacree, 2020).

2.13. Firebase

Firebase es una plataforma en la nube derivada de un proyecto que nació en el año 2011 como una base de datos en tiempo real y que, tras pasar a formar parte de Google en el año 2014, experimentó una evolución hacia un paquete completo de soluciones orientadas al desarrollo de aplicaciones web y móvil. Sus distintas soluciones o funcionalidades se distribuyen en cuatro grupos: compilación, lanzamiento y supervisión, analytics y participación. Además, cuenta con una serie de extensiones de terceros que amplían las funcionalidades de la plataforma. Así como Chatterjee et al. (2018) usan sendas funcionalidades de Firebase en su trabajo, en este proyecto se emplearán dos funcionalidades: Firebase Authentication, perteneciente al grupo de compilación, y Firebase Cloud Messaging, perteneciente al grupo de participación.

2.13.1. Firebase Authentication

El servicio Authentication gestiona de manera completa el acceso de usuarios. Permite habilitar el control de acceso usando correo electrónico y contraseña, teléfono, cuentas de Google, Facebook, Microsoft, Twitter, Yahoo, GitHub, Apple, Play Juegos y Game Center, además de permitir accesos anónimos. En el caso particular de acceso a través de correo electrónico gestiona la recuperación de contraseñas olvidadas y también se puede configurar para no usar contraseñas y usar vínculos enviados al correo electrónico con una tiempo de vida limitado.

2.13.2. Firebase Cloud Messaging

Este servicio permite enviar de manera masiva campañas de mensajes y notificaciones a usuarios registrados y monitorizar si han sido atendidas. Es un servicio muy potente que, tal y como afirmaron Mokar et al. (2019) en la conferencia internacional sobre ingeniería informática, de control, eléctrica y electrónica (ICCCEEE) del año 2019, permite la comunicación con aplicaciones y grupos de usuarios mediante el empleo del formato JSON conteniendo duplas entrada-valor.

Un ejemplo de construcción de un mensaje en formato JSON conteniendo varios pares entrada-valor sería así:

```
{
   "message": {
      "topic": "news",
      "notification": {
         "title": "Noticia de prueba",
         "body" : "Esto es una notificación de prueba"
      },
      "data": {
         "entrada 1": "valor 1",
         "entrada 2": "valor 2",
         "entrada 3": "valor 3",
          [...]
         "entrada n": "valor n",
      }
   }
}
```

Texto 3: Ejemplo de mensaje con formato JSON para enviar notificación a través de Firebase

CAPÍTULO 3: PLANIFICACIÓN TEMPORAL Y COSTES

3.1. Planificación temporal

Un condicionante para el desarrollo del sistema planteado en este trabajo es el límite de 300 horas al que se debe ajustar. Otra característica es que será desarrollado por una única persona. Así pues, se empleará un modelo lineal de etapas bien definidas similar al propuesto por el modelo en cascada pero sin producir la documentación y procesos de validación entre etapas que define formalmente dicho modelo (Origosa López, R. & Vázquez Ramiro, L., 2016). La Tabla 3 define la duración estimada de cada etapa.

Tabla 3: Detalle del cálculo de la duración del desarrollo del sistema de alarma

Fase	Duración (horas)
Análisis y definición de requisitos	
Análisis de mercado	4
Investigación documental	4
Análisis y definición de requisitos	8
Prototipos	
Creación de sketchs	4
Creación de las interfaces de usuario	8
Implementación	
Investigación tecnologías	40
Desarrollo aplicación centralita	32
Configuración del SO servidor	24
Desarrollo aplicación servidor	80
Desarrollo aplicación móvil	72
Pruebas	
Especificación de pruebas	8
Validación de pruebas	8
Extracción de conclusiones	8
TOTAL	**300**

La Figura 10 ilustra mediante un diagrama de Gantt la proyección sobre el calendario de las etapas anteriormente definidas. Dicha proyección ha sido configurada con jornadas de trabajo de 8 horas, cinco días laborables a la semana (de lunes a viernes) y se han marcado

como festivos los días reflejados en el Decreto 219/2021, de 22 de septiembre, del Consejo de Gobierno, por el que se establecen las fiestas laborales para el año 2022 en la CAM.

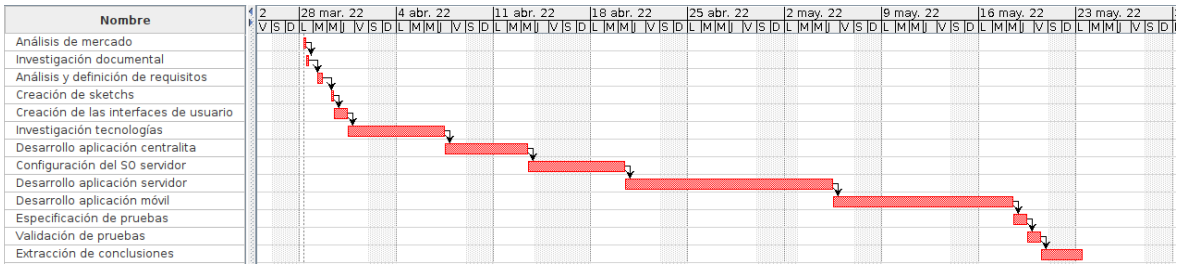


Figura 10: Diagrama de Gantt con la implantación del proyecto.
Elaborado con la herramienta ProjectLibre.

3.2. Costes

Para el desarrollo y prueba del sistema se ha realizado una instalación piloto en una vivienda que ha conllevado un gasto en materiales. En la Tabla 4 se puede ver con detalle el desglose de gastos realizado y el coste total el cual asciende a 401,86 euros.

Tabla 4: Estimación de coste de desarrollo del producto

Descripción	Coste unitario	Cantidad	Coste total
Caja seguridad grado 2 PAR-116	35,39 €	1	35,39 €
Fuente alimentación DRC-60A grado 2	34,30 €	1	34,30 €
Batería 12V/4.2AMP sin mantenimineto	11,00 €	1	11,00 €
Rollo de cable apantallado 6x0,18 100m	29,04 €	1	29,04 €
Kit 5 detectores apertura MC-38 grado 2	7,99 €	1	7,99 €
Detector Pyronix grado 2 KX10DTP	21,05 €	4	84,20 €
Cámara IP ONVIF visión nocturna 20m	62,45 €	1	62,45 €
Sirena interior grado 3 AS210N	13,23 €	1	13,23 €
Teclado-Lector Weigand 13.56 MHz	24,76 €	1	24,76 €
Arduino Mega 2560 Rev 3	42,35 €	1	42,35 €
Raspberry Pi 3 Model B+	35,15 €	1	35,15 €
Tarjeta microSD 128GB Class10 U3 V30	22,00 €	1	22,00 €
		TOTAL	**401,86 €**

Para la estimación de costes hay que tener en cuenta que todo el software utilizado en el proyecto es libre y gratuito (sistemas operativos, entornos de desarrollo, librerías, imágenes, etcétera) y el producto desarrollado también se licencia como software libre bajo la licencia pública general GNU en su versión 3 (Free Software Foundation, 2007), por lo que no se generarán costes sobre estas partidas. Igualmente, la instalación del prototipo fue realizada de manera gratuita por el autor.

CAPÍTULO 4: DESARROLLO

4.1. Requisitos

Los sistemas de alarma no son sistemas que requieran habilidades especiales por parte de sus usuarios, estos deben limitarse a armar y desarmar el sistema a través de un control de acceso que tiene un teclado numérico y un lector de etiquetas RFID. El sistema que se propone en este trabajo ofrece adicionalmente una herramienta tecnológica (una app) al usuario para interactuar con el sistema que le permitirá armar y desarmar el sistema, visualizar imágenes captadas por cámaras, recibir notificaciones push y consultar el histórico de eventos.

Teniendo en cuenta los objetivos marcados y tras estudiar el funcionamiento de los sistemas de alarma indicados por la normativa vigente, se puede enumerar un conjunto de requisitos necesarios para crear un marco de desarrollo para el sistema. Los requisitos funcionales implementarán los casos de uso mientras que los requisitos no funcionales definirán cómo se debe comportar el sistema.

4.1.1. Requisitos funcionales

RF01 - Armado del sistema de alarma

Cualquier persona podrá armar la alarma marcando en el teclado del control de acceso un código numérico preestablecido por la centralita de alarma. Los usuarios, además, podrán armar la alarma usando la app. El armado de la alarma con la app tendrá efecto inmediato, mientras que el armado a través de código tendrá un retardo de 20 segundos.

RF02 - Desarmado del sistema de alarma

Todos los usuarios podrán definir un código numérico para desarmar la alarma, además podrán desarmar la alarma a través de la app y mediante la aproximación de una etiqueta RFID que haya sido previamente asociada al usuario.

RF03 - Rol de usuario administrador

Habrá al menos un usuario con rol administrador. Este rol permitirá añadir y eliminar usuarios y cámaras y permitirá configurar usuarios, cámaras y sensores.

RF04 - Registro e identificación de los usuarios

Los usuarios se registrarán usando una dirección de correo electrónico única y personal para cada usuario. Dicha cuenta de correo electrónico será la empleada para acceder a la app.

RF05 - Inicio manual de una ráfaga de capturas de una cámara

Los usuarios podrán solicitar a través de la app una captura de imágenes de una cámara dada de alta en el sistema.

RF06 - Consulta de imágenes capturadas

Todos los usuarios podrán visualizar a través de la app las imágenes capturadas por las cámaras, que se organizarán por huellas de tiempo (fecha y hora).

RF07 - Historial de eventos

Los usuarios podrán visualizar a través de la app un histórico de los últimos enésimos eventos.

4.1.2. Requisitos no funcionales

RNF01 - No uso de radiofrecuencias

El sistema de alarma no debe utilizar radiofrecuencias para la comunicación de los distintos elementos que lo componen.

RNF02 - Aviso acústico

La alarma debe emitir un aviso acústico local al dispararse.

RNF03 - Funcionamiento *stand-alone* de la centralita

La centralita debe ser capaz de funcionar en modo *stand-alone* para asegurar el funcionamiento de la alarma ante una caída de otros servicios.

RNF04 - Notificación push en caso de disparo

El sistema debe emitir una notificación push a la app de todos los usuarios registrados ante cualquier disparo.

RNF05 - Inicio automático de una ráfaga de capturas de una cámara

Cuando el sistema esté armado, un detector se active y el detector activado tenga asociada una cámara, ésta deberá capturar una ráfaga de al menos 3 imágenes en un espacio de tiempo de 5 segundos.

RNF06 - Disparo diferido. Predisparo

El sistema debe permitir configurar el disparo diferido (20 segundos) en cada sensor. Este requisito tiene el único objetivo de dar tiempo a desarmar el sistema mediante la introducción de un código numérico o la aproximación de una etiqueta RFID.

RNF07 - Aviso acústico de predisparo

Se emitirá un aviso acústico consistente en pitidos intermitentes (por ejemplo, 500 ms pitando, 500 ms silencio) durante el periodo de predisparo y los últimos 5 segundos se acelerará el ritmo de los pitidos (por ejemplo, 250 ms pitando, 250 ms silencio).

RNF08 - Armado diferido. Prearmado

Cuando el sistema sea armado mediante la introducción de un código numérico se debe diferir 20 segundos su armado efectivo.

RNF09 - Aviso acústico de prearmado

Se emitirá un aviso acústico consistente en pitidos intermitentes (por ejemplo, 500 ms pitando, 500 ms silencio) durante el periodo de prearmado y los últimos 5 segundos se acelerará el ritmo de los pitidos (por ejemplo, 250 ms pitando, 250 ms silencio).

RNF10 - Funcionamiento tras interrupción de suministro de energía eléctrica

La centralita de alarma debe funcionar localmente ante un corte de suministro de energía eléctrica en la vivienda al menos durante 12 horas.

RNF11 - Cantidad de sensores

La centralita de alarma debe manejar un total de 11 sensores.

RNF12 - Atención de peticiones de la app

Se atenderán las peticiones de los distintos clientes (app) mediante un servicio RMI o similar.

RNF13 - Concurrencia de clientes

El servidor deberá manejar la conexión de más de un cliente al mismo tiempo.

RNF14 - Versión mínima del Sistema Operativo Android

Se diseñará la app para trabajar en dispositivos con Sistema Operativo Android 5.1 (nivel de API 22) o superior, de esta forma se dará alcance al 97,3% (fuente: Android Studio) de los dispositivos Android. Como el 70,97% (fuente: statcounter.com) del mercado de smartphones usa Android, se dará alcance al 69,05% de smartphones.

RNF15 - Raspberry Pi

Se utilizará una Raspberry Pi 3 (modelo B o B+) para tener un equipo con servicio 24/7[1] en el que hospedar el motor de la base de datos y el servidor del sistema de alarma que hará de interfaz entre la centralita y la aplicación de los usuarios.

RNF16 - Arduino

Se usará una placa Arduino Mega 2560 para implementar la centralita de alarma.

RNF17 - Sistema Operativo Raspberry Pi

Se hará funcionar la placa Raspberry Pi con el Sistema Operativo Raspberry Pi conocido como *Raspbian GNU/Linux*.

RNF18 - Servidor alarma

Se implementará una aplicación que dé servicio 24/7 para hacer de interfaz entre los distintos elementos que componen el sistema de alarma. Este servicio se hospedará en la Raspberry Pi.

RNF19 - Motor base de datos SQL

Se instalará un motor de base de datos relacional compatible con SQL en la Raspberry Pi para el almacenamiento de usuarios, eventos, cámaras, etcétera.

RNF20 *- Open Java Development Kit*

Se instalará en la Raspberry Pi el OpenJDK más actual por defecto para poder lanzar la aplicación del servidor.

RNF21 - Conexión a Internet

Se deberá contar con conexión a Internet con medio de transmisión guiado en la vivienda.

1 24/7 es un término proveniente de la expresión "24 horas al día, 7 días a la semana" que es empleado para referirse a servicios que no tienen interrupción de disponibilidad.

4.2. Análisis

Se identifican dos actores, por un lado el usuario normal y por otro lado el usuario administrador que extiende las funciones de un usuario normal.

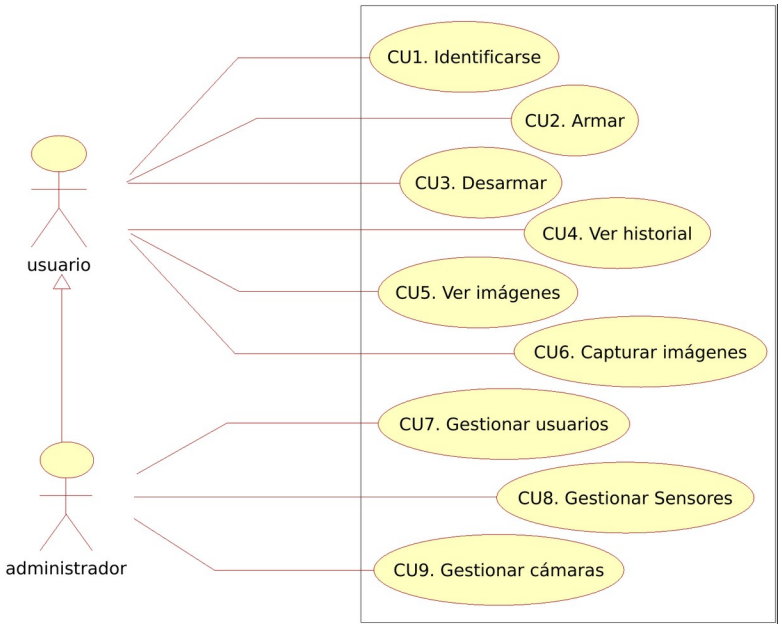


Figura 11: Casos de uso. Elaborado con la herramienta Umbrello

Los casos de uso mostrados en la Figura 11 se desarrollan con detalle en el “Anexo A2. Boceto y flujo de pantallas de la app”.

4.3. Diseño

4.3.1. Arquitectura del sistema de alarma

Desde un punto de vista global se pueden enumerar los principales elementos que componen el sistema de alarma: la app, el servidor y la centralita. Sin embargo, para proyectar el diseño de la arquitectura del sistema se deben incluir otros elementos: la base de datos, las cámaras, el servicio Firebase de Google y las redes TCP/IP utilizadas.

El sistema de alarma hará uso de la Home Area Network (HAN) de la vivienda. Las HAN se caracterizan por ser redes de área local (LAN) que incorporan una puerta de enlace que las conectan a la red Internet. Normalmente la puerta de enlace está embebida en un dispositivo que proporciona la empresa proveedora de servicios de Internet (ISP), este dispositivo, además de contener una puerta de enlace contiene otros elementos que se usarán como el servicio NAT (Network Adress Translation) para redireccionar un puerto de escucha desde la IP pública en Internet hasta la IP privada del servidor en la HAN.

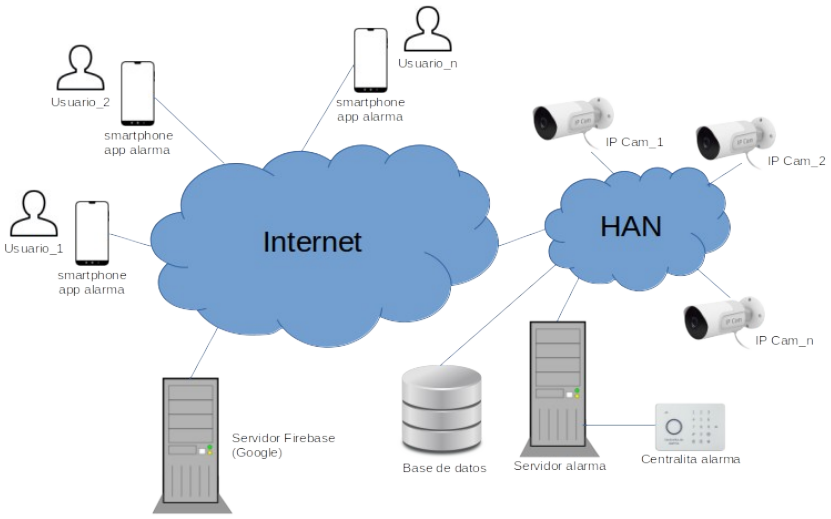


Figura 12: Arquitectura del sistema de alarma

Tal y como ilustra la Figura 12, la centralita y el servidor tienen una comunicación punto a punto y el resto de elementos se comunican utilizando la red TCP/IP. Como el servidor debe ser visible desde Internet, además del servicio NAT mencionado, se deberá contar con un mecanismo de direccionamiento (IP fija, DNS, DDNS, etcétera).

4.3.2. Servidor

El servidor tendrá dos interfaces de conexión, la interfaz RS-232 con una conexión *null modem* la cual define una conexión peer-to-peer (P2P) que será usada para comunicarse con la centralita, y la conexión de red Ethernet[2] mediante un medio de transmisión guiado (par trenzado) que conectará el servidor a la red TCP/IP.

La conexión a la red Ethernet tendrá diversos usos, ofrecerá un servicio RMI a través de un puerto determinado que se usará para que los usuarios envíen y/o soliciten información directamente al servidor desde la app; también el servidor enviará mensajes POST (HTTP) al servidor Firebase para lanzar datos y notificaciones a la app de los usuarios; otro uso será acceder al servicio de red del motor de la base de datos relacional SQL diseñada en el apartado “4.3.4. Base de datos relacional”; y un cuarto uso será el acceso al servicio de la cámara o las cámaras IP que pudieran instalarse.

Las implementaciones de las funciones del servidor mencionadas tendrán un componente bloqueante por lo que deberán ser gestionadas mediante subprocesos independientes (hilos o threads, en inglés) con la intención de atender a todas ellas de manera concurrente. Principalmente se pueden destacar cuatro actividades bloqueantes, dos de ellas siempre estarán activas:

- **Servicio RMI**. Bloqueante ante la espera de conexiones de clientes.
- **Recepción datos RS-232**. Bloqueante ante la espera de recepción de datos de la centralita.

Y otras dos actividades se crearán cuando se necesiten:

- **Envío de mensajes POST al servidor Firebase**. Bloqueante ante la espera de la respuesta HTTP.
- **Captura de imágenes de cámara IP**. Bloqueante ante la espera de recepción de imágenes.

2 Con el término Ethernet se hace referencia a las distintas versiones del estándar IEEE 802.3

4.3.2.1. Comunicaciones orientadas a la conexión

Las comunicaciones Ethernet, principalmente el servicio RMI, el envío de mensajes a Firebase y la captura de imágenes de cámaras IP, estarán basadas en flujos de datos orientados a la conexión extremo a extremo, de este modo se maximiza la fiabilidad de las comunicaciones gracias al envío ordenado de paquetes y acuse de recibo que ofrece el protocolo TCP.

4.3.2.2. Servicio RMI

El servicio RMI se diseña como un sistema con un servidor complejo actuando como intermediario entre varios clientes, este sistema consigue manejar la concurrencia de varios clientes manteniendo un servicio de escucha del puerto donde se presta el servicio y pasando las distintas conexiones a subprocesos creados para atender las peticiones de los clientes. La Figura 13 ilustra el diseño mencionado empleando los componentes Socket y ServerSocket. En la figura destaca el carácter bloqueante de las acciones Aceptar del componente ServerSocket y Recibir del componente Socket.

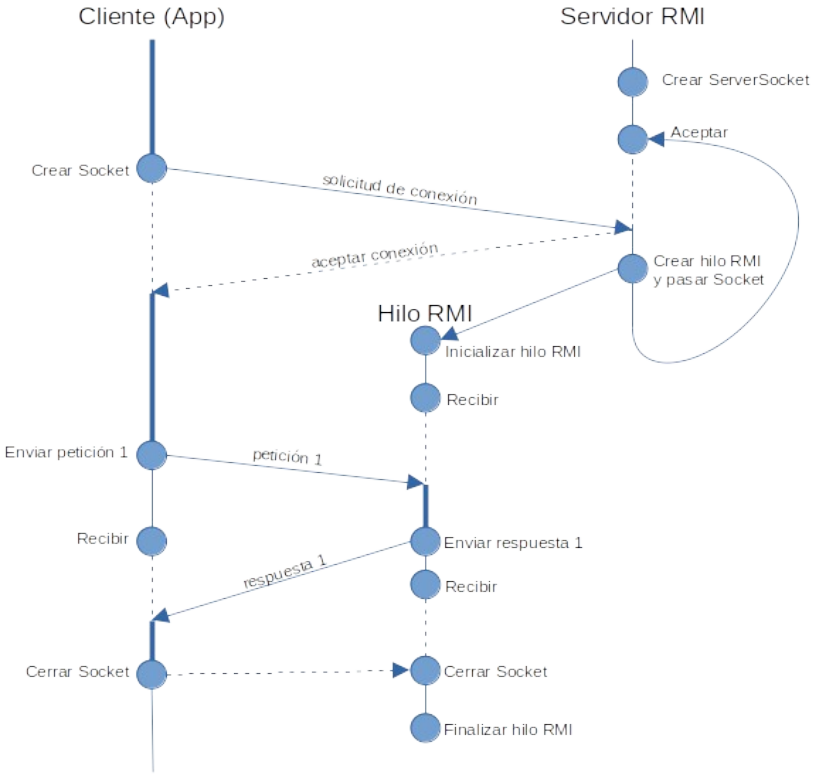


Figura 13: Diagrama de eventos del servicio RMI

4.3.2.3. Comunicación centralita-servidor

La comunicación entre el servidor y la centralita consistirá en el envío de mensajes asíncronos. El diseño del mecanismo de atención de mensajes asíncronos en el lado de la centralita es sencillo puesto que ésta funcionará con un mecanismo de polling que llamará a una función de atención de mensajes en cada vuelta, si hay mensaje lo atenderá. El diseño en el lado del servidor consistirá en un proceso independiente que recibirá los datos del puerto serie, los validará y los añadirá a una cola con prioridad. El uso de una cola con prioridad se justifica al existir mensajes que requieren atención preferente frente a otros, por ejemplo, un mensaje que indica un salto de alarma tendrá más preferencia que otros.

Por otro lado, para atender los mensajes de la cola con prioridad debe implementarse otro servicio el cual supervisará continuamente la cola y, ante la presencia de mensajes, sacará el primero eliminándolo de la cola y lo atenderá.

4.3.3. Centralita

4.3.3.1. Técnica polling

La centralita de la alarma basará su funcionamiento en la técnica polling que viene impuesta en gran medida por el funcionamiento planteado por los productos Arduino. De este modo, la clase principal ejecutará primero la función setup() y a continuación la función loop() continuamente. Es en la función loop() en la que se incluye el chequeo continuo de los distintos elementos implementando así la técnica polling. Toda clase susceptible de ser chequeada en este proceso implementará una función denominada **check()** la cual será llamada en cada ciclo.

4.3.3.2. Estado de la alarma

El estado de la centralita podrá ser *Desarmado*, *Armado* o *Disparado*, ahora bien, para dar tiempo a salir de la vivienda cuando se arme a través de la introducción de un código en el teclado o para que de tiempo a desarmar el sistema al entrar en la vivienda sin que se dispare la alarma, existirán los estados *Prearmado* y *Predisparado*. Estos dos últimos estados se usarán para indicar acústicamente que el sistema se armará o se disparará. Normalmente se emite unos pitidos con una frecuencia más o menos tranquila seguidos de unos últimos pitidos que aumentan de frecuencia, por ello, cada estado será representado

por dos estados: *Prearmado_inicial*, *Prearmado_final*, *Predisparado_inicial* y *Predisparado_final*, los iniciales con pitidos normales y los finales, más acelerados.

Desde el estado *Desarmado* se puede armar el sistema y pasar directamente al estado *Armado*, o prearmar el sistema y pasar al estado *Prearmado_inicial* que pasará al estado *Prearmado_final* tras 15 segundos, éste a su vez, tras 5 segundos, pasará al estado *Armado*.

Desde el estado *Armado* se puede disparar el sistema y pasar directamente al estado *Disparado*, o predisparar el sistema y pasar al estado *Predisparado_inicial* que pasará al estado *Predisparado_final* tras 15 segundos, éste a su vez, tras 5 segundos, pasará al estado *Disparado*.

Desde cualquier estado se puede desarmar el sistema y pasar directamente al estado *Desarmado*.

Todo el comportamiento descrito en los párrafos anteriores se puede representar mediante un autómata de Moore tal y como ilustra la Figura 14.

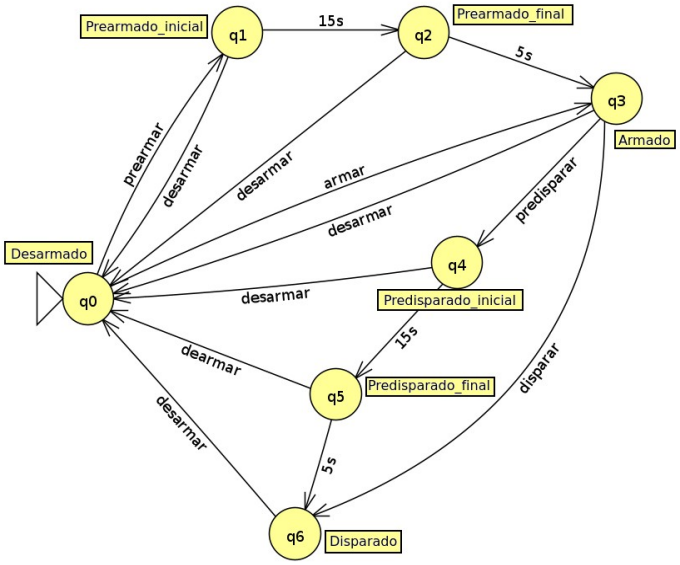


Figura 14: Representación de los estados de la alarma mediante un autómata de Moore. Elaborado con la herramienta JFLAP

4.3.3.3. Indicación luminosa del estado

El control de acceso hará lucir un indicador luminoso en función del estado de la alarma: alternará entre rojo y verde a una velocidad lenta (por ejemplo, cada segundo) cuando el

estado de la alarma sea Armado; la velocidad será rápida (por ejemplo, cada 200 milisegundos) cuando el estado sea Disparado; y rojo continuo en el resto de los estados.

4.3.3.4. Armado del sistema

El sistema de la centralita se armará introduciendo el código 159. Este código representa la primera diagonal si se tiene en cuenta el sentido de lectura occidental, por lo que se presume fácil de recordar. De este modo, cualquier persona podrá armar el sistema.

4.3.3.5. Envío de mensajes al servidor

La centralita enviará al servidor un mensaje cada vez que inicie el proceso de polling, cada vez que el estado de la centralita cambie, cada vez que un sensor provoque un disparo o predisparo del sistema, cada vez que se introduzca un código numérico diferente al código de armado y cada vez que se lea una etiqueta RFID.

4.3.4. Base de datos relacional

Se ha realizado un diseño basado en el modelo Entidad-Relación (Lorenzo Bragado, C., 2013) en el que se han tomado las siguientes decisiones basadas en los requisitos planteados.

Se ha decidido plantear las entidades Usuario, Evento, Sensor y Cámara para almacenar, por un lado, la configuración de la centralita y, por otro lado, la gestión de usuarios que se usará para limitar el acceso de estos al sistema.

Solamente existe una relación entre las entidades Sensor y Cámara. Esta relación es de tipo N:M ya que una cámara puede ser disparadas por uno o más sensores y un sensor puede disparar una o más cámaras.

Para la entidad Usuario se ha decidido emplear la dirección de correo electrónico (propiedad Email) como clave primaria puesto que hoy día se entiende que cada persona puede conseguir una dirección de correo electrónico unívoca para sí misma, no obstante, la dirección de correo electrónico puede no existir en el supuesto de dar de alta un usuario que no vaya a utilizar la app pero sí vaya a utilizar un llavero RFID o un código numérico para desarmar el sistema.

La clave principal de la entidad Evento será un número incremental automático puesto que, aunque pueda parecer que la propiedad Timestamp es unívoca, podría darse el caso de

recibir dos eventos generados con la misma marca o sello de tiempo, que es lo que refleja la propiedad Timestamp.

La clave principal de la entidad Sensor es un número identificador (propiedad Id) que se corresponderá con el número identificador unívoco asignado en la centralita, por este motivo los sensores deberán inicializarse junto con la creación de la estructura de la base de datos.

La clave principal de la entidad Cámara será la propiedad Alias puesto que el alias asignado a cada cámara deberá ser unívoco en el sistema.

Así mismo, las propiedades Alias de las entidades Usuario y Sensor deben tener carácter unívoco puesto que este campo es el usado en la interfaz de usuario.

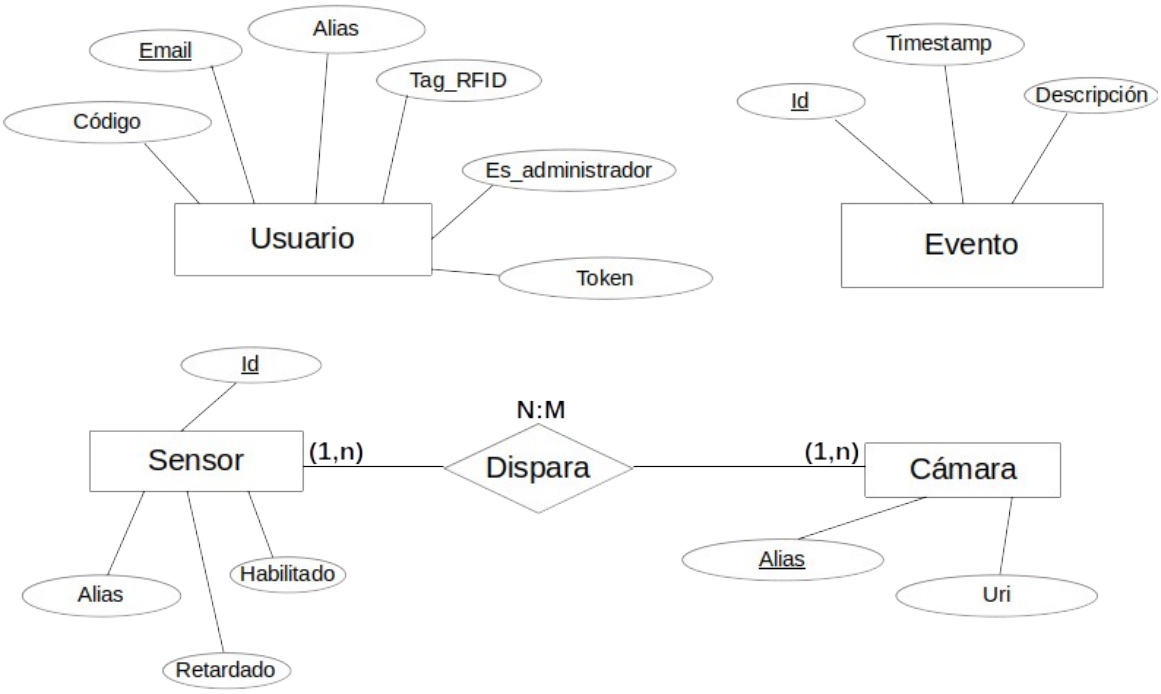


Figura 15: Diseño de base de datos relacional basado en modelo entidad-relación

Entre las entidades Sensor y Cámara existe una cardinalidad N:M que implicará crear una nueva tabla cuya clave primaria será la concatenación de las claves primarias de las tablas relacionadas (Sensor y Cámara).

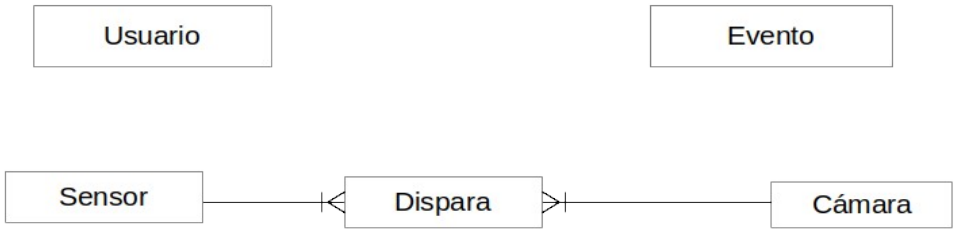


Figura 16: Diseño lógico de la base de datos

En la Figura 16 se puede observar el diseño lógico derivado del diseño inicial en el que se puede apreciar cómo la cardinalidad N:M existente entre las entidades Sensor y Cámara se ha convertido en la entidad Dispara.

De los diseños anteriores obtenemos las tablas que se enumeran a continuación y que serán objeto de implementación en el epígrafe siguiente.

Usuario (Email, Alias, Es_administrador, Token, Tag_RFID, Código)

Evento (Id, Timestamp, Descripción)

Sensor (Id, Alias, Habilitado, Retardado)

Cámara (Alias, Uri)

Dispara (Id_Sensor_Dispara (FK), Alias_Camara_Disparada (FK))

4.3.5. Aplicación para smartphone Android

4.3.5.1. Validación de usuario

Se incluirá un mecanismo para validación de usuario basado en dos verificaciones, primero por la plataforma Firebase y posteriormente por el servidor de alarma que cotejará la dirección de correo electrónico validada por Firebase con las guardadas en la base de datos. Una vez validado el usuario, la app enviará al servidor de alarma el token que identifica la app del usuario para el envío de mensajes y notificaciones desde la plataforma Firebase.

4.3.5.2. Interfaz de usuario

Se inicia el diseño de la interfaz de usuario con el esquema de navegación ilustrado en la Figura 17 que sirve como hilo conductor para, junto con los requisitos, esbozar un boceto de la app y los flujos de navegación.

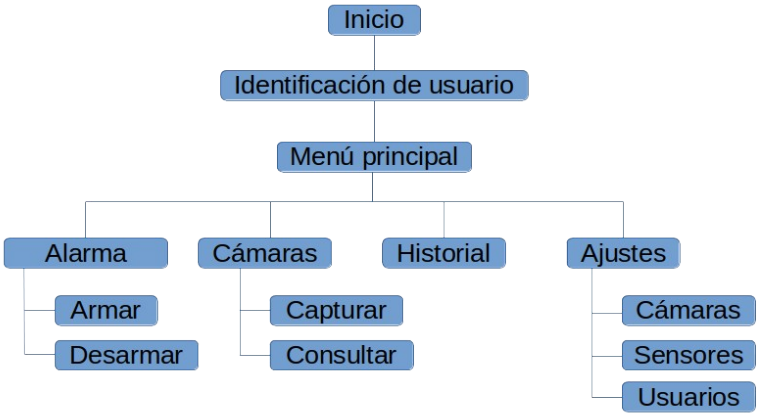


Figura 17: Mapa de navegación de la app

El boceto de flujo de pantallas derivado del esquema anterior se encuentra ilustrado en el "Anexo A2. Boceto y flujo de pantallas de la app" y cubre los requisitos funcionales establecidos. Existen algunos requisitos funcionales que deben completarse mediante el uso del control de acceso (teclado y lector de etiquetas RFID) que será implementado en el código del servidor y de la centralita.

4.4. Implementación

4.4.1. Instalación y configuración de software

El sistema operativo instalado en la placa Raspberry Pi es el "Raspberry Pi OS". Se trata de un sistema operativo GNU/Linux de 32 bits de la familia Debian con versión kernel 5.10. Tras su instalación se han realizado las siguientes instalaciones y configuraciones:

- Puesta al día de paquetes, versiones e instalación de los mismos
  ```
  sudo apt update && sudo apt upgrade
  ```
- Habilitación de servicio ssh y habilitación del puerto serie
  ```
  sudo raspi-config
  ```
- Instalación de la versión OpenJDK más actual (11.0.14)
  ```
  sudo apt install default-jdk
  ```
- Instalación y primera configuración del motor de base de datos relacional MariaDB
  ```
  sudo apt install mariadb-server mariadb-client
  sudo mysql_secure_installation
  ```
- Instalación de la herramienta conspy
  ```
  sudo apt install conspy
  ```

- Configuración de IP (v4) estática, puerta de enlace y servidores DNS mediante la edición del archivo `/etc/dhcpcd.conf`

```
sudo vi /etc/dhcpcd.conf
    interface eth0
    static ip_address=192.168.1.225/24
    static routers=192.168.1.1
    static domain_name_servers=1.1.1.1 1.0.0.1
```

- Creación de la estructura de directorios necesaria para la aplicación

```
sudo mkdir -p /usr/local/bin/servidoralarma/lib/
```

- Creación y configuración del servicio para su ejecución automática

```
sudo vi /lib/systemd/system/servicioalarma.service
  [Unit]
     Description=Servicio Alarma
     After=multi-user.target
     After=getty.service
  [Service]
     Type=idle
     WorkingDirectory=/usr/local/bin/servidoralarma/
     ExecStart=/usr/bin/java -jar /usr/local/bin/servidoralarma/ServidorAlarma.jar
     StandardInput=tty
     StandardOutput=tty
     StandarError=tty
     TTYPath=/dev/tty20
     RemainAfterExit=yes
  [Install]
     WantedBy=multi-user.target
sudo chmod 644 /lib/systemd/system/servicioalarma.service
sudo systemctl daemon-reload
sudo systemctl enable servicioalarma.service
```

Para acceder a la salida y entrada estándar con la que trabajará el software servidor de alarma bastará con acceder a través del servicio ssh y ejecutar el comando `sudo conspy 20`.

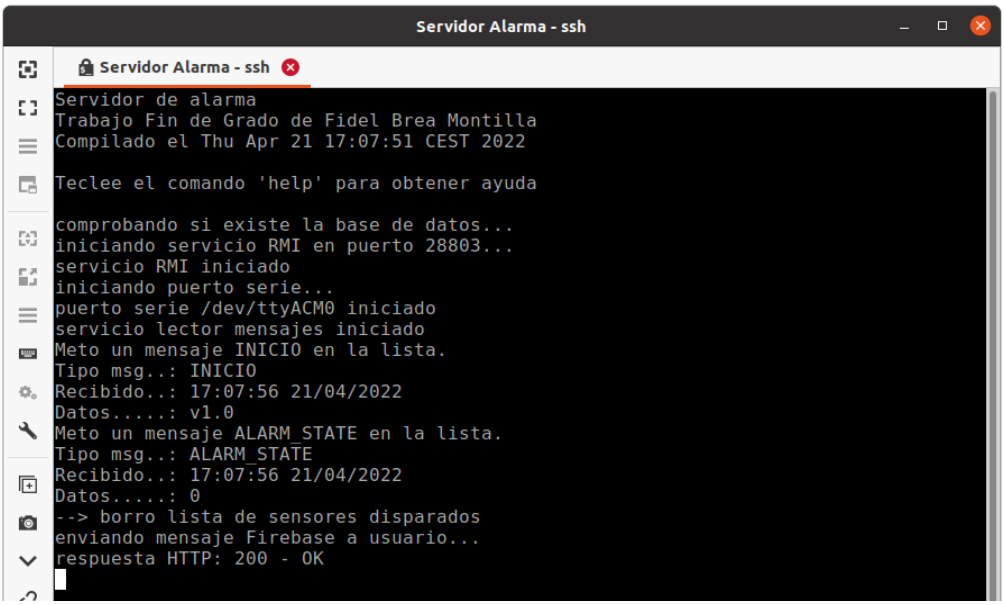


Figura 18: Acceso a los flujos de salida y entrada del servidor a través de ssh

La Figura 18 muestra el aspecto de una conexión a los flujos de entrada y salida del servicio mediante el método mencionado.

4.4.2. Protocolo centralita – servidor

El intercambio de mensajes entre el servidor y la centralita se basa en mensajes ligeros con formato JSON que contendrán las etiquetas *tipo* y *datos*. Los distintos mensajes que mandará la centralita al servidor se muestran en la Tabla 5 y los mensajes que mandará el servidor a la centralita se recogen en la Tabla 6. Ambas tablas se localizan en el "Anexo A3. Mensajes centralita-servidor". Hay que destacar la información que se envía a la centralita cada vez que se arma o prearma permite a la centralita operar de manera autónoma evitando la necesidad de contrastar con el servidor los datos de códigos numéricos de desarmado o informaciones de etiquetas RFID. Un ejemplo de la información enviada a la centralita se puede ver en el Texto 6 del "Anexo A3. Mensajes centralita-servidor".

4.4.3. Servidor

La clase **Main** contiene la función principal en la que primeramente se comprueba la existencia de la base de datos y, de no existir, la crea con el contenido de los archivos *script.sql* e *insert.sql*. A continuación instancia la clase Servidor e inicializa los servicios RMI, puerto serie y lectura de mensajes tal y como se diseñó en el epígrafe "4.3.2.3. Comunicación centralita-servidor". Finalmente queda a la espera de la lectura de una línea de texto proveniente de la entrada estándar, con ello se implementa un funcionamiento de consola de comandos que sirve al desarrollador para la realización de pruebas del sistema y obtención de diversas informaciones.

La clase **Servidor** es la clase principal del sistema y se usa como contexto del mismo dando a los objetos que la reciben acceso a variables y otros objetos instanciados por ella.

La clase **DatosDB** se instancia desde la clase Servidor y contiene todas las funciones que acceden a la base de datos. Esto se ha decidido así para centralizar en un único lugar estas acciones y facilitar el mantenimiento del software.

La clase **ManejadorJson** se instancia desde la clase Servidor y contiene todas las funciones que manejan los mensajes con formato JSON, incluyendo el envío de estos.

La clase **TipoMensaje** es una enumeración de los tipos de mensajes que pueden llegar desde la centralita y que se recogen en la Tabla 5. Solamente los mensajes que formen parte de esta enumeración serán aceptados e introducidos en la cola de mensajes.

Las clases **Usuario**, **Sensor**, **Disparo** y **Camara** ayudan a manejar los datos provenientes de la base de datos, así como la clase **Mensaje** se usa para manejar los mensajes provenientes de la centralita.

La clase **CapturadorImagenesThread** extiende la clase Thread y es la encargada de capturar una ráfaga de imágenes de una cámara particular.

La clase **MensajeFirebaseThread** extiende la clase Thread para enviar un mensaje al servidor Firebase de Google y esperar la respuesta del protocolo HTTP.

La clase **ServicioLectorMensajesThread** extiende la clase Thread y está esperando a que haya mensajes en la cola con prioridad. Cuando hay mensajes, los va sacando de la cola y atendiendo.

La clase **PuertoSerieEscuchador** implementa la interfaz SerialPortDataListener de la librería jSerialComm para atender los datos cuando llegar al puerto serie. Una vez que detecta que hay un mensaje JSON lo introduce en la cola con prioridad.

La clase **ServicioRmiEscuchador** implementa la interfaz IserverListener de la librería LipeRMI para atender los eventos de conexión y desconexión de clientes.

La clase **ServicioRmiImp** implementa la interfaz **ServicioRmiInt** compartida con todos los clientes, en el caso de este proyecto esta interfaz se comprate con la app.

Todo el código se encuentra disponible en el repositorio *home-alarm* de github.com.

4.4.3.1. Servicio RMI

La implementación del servicio RMI diseñado en el apartado “4.3.2.2. Servicio RMI” se encuentra en la clase Server del proyecto LipeRMI cuyo código ha sido revisado e incorporado al proyecto del servidor. La función bind de la clase Server se recoge en el

“Anexo A4. Servicio RMI”. Es la encargada de iniciar el servicio RMI tal y como ha sido diseñado.

4.4.3.2. Comunicación con el servidor Firebase

La comunicación con el servicio Firebase Cloud Messaging se basa en el envío de mensajes HTTP al servidor https://fcm.googleapis.com/fcm/send (nótese el uso del protocolo HTTPS) con una carga de datos que contiene un mensaje con formato JSON (Content-type: application/json). En la cabecera HTTP se incluiye una clave que proporciona Google desde su plataforma Firebase. El mensaje JSON se conforma según las pautas dadas por Google en su documentación on-line.

El “Anexo A5. Comunicación con Firebase Cloud Messaging” recoge la función encargada de conformar el mensaje HTTP y el mensaje JSON y enviarlo al servidor Firebase tantas veces como tokens haya en la lista *tokens* usando un OutputStreamWriter sobre un objeto HttpUrlConnection. Así mismo, tras su envío, se espera la respuesta para obtener el código y descripción de la respuesta HTTP.

4.4.3.3. Comunicación con la centralita

La comunicación entre el servidor y la centralita se realiza mediante una comunicación serie RS-232. Existe una limitación en las fuentes del proyecto Arduino que se debe tener en cuenta. Esta limitación consiste en el tamaño del buffer de entrada del puerto serie que en las fuentes de Arduino está establecido en 64 bytes. Esto obliga a que toda comunicación saliente hacia la centralita debe realizarse en bloques no superiores a 64 bytes y dar tiempo entre envíos a que la centralita los procese.

```
public static void enviarJSON(SerialPort s, String json) {
     String[] mensajeDividido = json.split("(?<=\\G.{60})");
     for (String parte : mensajeDividido) {
          s.writeBytes(parte.getBytes(), parte.length());
          s.flushIOBuffers();
          try {
               Thread.sleep(200);
          } catch (InterruptedException e) {}
     }
}
```

Texto 4: Envío de datos a la centralita

En el Texto 4 se puede observar cómo se divide la cadena de texto contenida en la variable *json* mediante el uso de la expresión regular `(?<=\\G.{60})` y que son enviados

en orden con pausas de 200 milisegundos que aseguran la recepción y vaciado del buffer de entrada en la centralita.

4.4.3.4. Comunicación con cámaras IP

La comunicación con las cámaras IP dependerá del protocolo que se haya definido al darla de alta. Si se emplea el protocolo HTTP se limitará a capturar un número determinado de veces la imagen disponible en la URI esperando un segundo entre capturas. Si se emplea el protocolo RTSP se llama al comando ffmpeg para que conecte e inicialice el servicio de streaming y capture una ráfaga de imágenes cada segundo.

El nombre que recibe cada archivo de imagen tiene el formato *yyyyMMddHHmmss.jpg*, es decir, año, mes, día, hora, minutos y segundos. De este modo, se puede asociar la captura a un momento concreto tal y como exige la Ley de cara a probar un allanamiento.

4.4.3.5. Fuentes de terceros utilizadas

El proyecto servidor de alarma se ha implementado en lenguaje Java y utiliza cuatro librerías externas: **jSerialComm** versión 2.9.1 (Hedgecock, W., 2022) para la comunicación serie, **mysql-connector-java** versión 8.0.28 (Oracle Corporation, 2022) para la comunicación con el motor SQL, **json-simple** versión 1.1.1 (Fang, Y., 2012) para el manejo de mensaje con formato JSON y **lipeRMI** versión 0.4.2 (Santos Andrade, F., 2006) para la implementación del servicio RMI. Estas fuentes se encuentran bajo las licencias GNU General Public License versión 3 (jSerialComm), GNU General Public License versión 2.1 (mysql-connector-java y lipeRMI) y la Apache License versión 2 (json-simple). Dado que el software creado en este proyecto se licencia bajo la GNU General Public License versión 3, no se genera ningún conflicto derivado de la reutilización y distribución de las librerías.

4.4.4. Centralita

En la Figura 19 se ha representado con detalle la implementación del proyecto de la centralita usando el lenguaje gráfico UML.

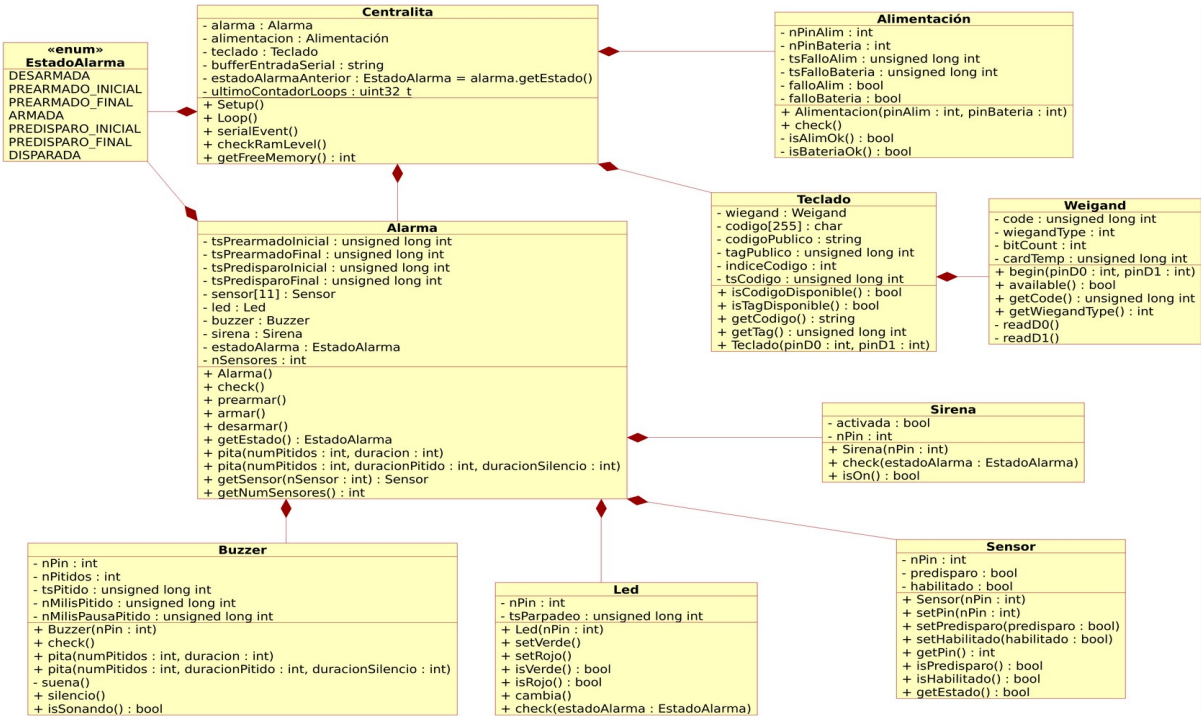


Figura 19: Diagrama de clases de la centralita. Elaborado con la herramienta Umbrello

La clase **Centralita** es la clase principal del proyecto en la cual están contenidas las funciones setup() y loop(). Esta clase realiza además el manejo de mensajes procedentes del puerto serie (comunicación servidor-centralita) e introduce un control de memoria RAM para reiniciar la placa si el nivel de memoria RAM es muy bajo y el estado de la alarma es Desarmado. Desde esta clase se instancian las clases Alarma, Alimentación y Teclado. Las tres clases implementan la función check() para su inclusión en el mecanismo de polling.

La clase **Alarma** se encarga de establecer el estado de la alarma e indicar al usuario mediante pitidos de un buzzer e iluminación de un led R/V (Rojo/Verde), los estados de la misma. Desde esta clase se instancian las clases Sensor, Sirena, Buzzer y Led. La clase Sensor se instancia en un array de dimensiones establecidas por los requisitos (cantidad de sensores), inicializa las entradas como INPUT_PULLUP y vigila el estado de cada uno de ellos si el estado de la alarma es Armado y el sensor está habilitado. Es aquí donde se producirán los predisparos y/o disparos de la alarma. Las clases Sirena, Buzzer y Led

implementan la función check() para su inclusión en el mecanismo de polling, por lo que se les invoca desde la función check() de la clase Alarma.

La clase **Sirena** se encarga de activar o desactivar la salida que acciona la sirena en función del estado de la alarma.

La clase **Buzzer** se encarga de emitir pitidos con los intervalos de tiempo que se le hayan pasado.

La clase **Led** se encarga de hacer parpadear el led R/V con los intervalos de tiempo que se le hayan pasado o dejarlo fijo en uno de los dos colores.

La clase **Teclado** informa mediante un par de banderas de la disponibilidad para su lectura de un código introducido por el usuario en el teclado del control de entrada o de un número de serie de una etiqueta RFID leída por el control de entrada. Se apoya en la clase **Wiegand** que se encarga de leer las señales Data0 y Data1 provenientes del control de entrada.

La clase **Alimentacion** inicializa dos entradas como INPUT_PULLUP y vigila las señales de presencia de alimentación y batería correcta provenientes de la fuente de alimentación.

Todo el código se encuentra disponible en el repositorio home-alarm de github.com.

4.4.4.1. Fuentes de terceros utilizadas

Para la implementación del diseño de la centralita se ha reutilizado el código del proyecto "**Wiegand protocol library for arduino**" desarrollado J.P. Liew (2020) y difundido por la organización MonkeyBoard para la interpretación de los códigos provenientes del control de entrada. El código de este proyecto se encuentra modificado en la clase Weigand.

También se ha incluido la biblioteca **ArduinoJson** versión 6.19.4 elaborada por Benoît Blanchon (2022) para el manejo de mensajes con formato JSON.

El código de estos proyectos se encuentran respectivamente bajo las licencias GNU Lesser General Public License versión 2.1 y la MIT License, por lo que no se genera ningún conflicto derivado de la reutilización y distribución del código.

4.4.4.2. Conexionado del pineado de la placa Arduino

La asignación de pines se puede ver en el "Anexo A6. Conexionado de elementos a la placa Arduino MEGA". Hay que recordar que los detectores de apertura de puertas y ventanas, los detectores de movimiento y los tampers se categorizan todos como sensores puesto que todos ellos proporcionan un contacto libre de potencial normalmente cerrado que durante el estado Armado de la alarma permanece cerrado para indicar normalidad y se abre para indicar una violación del elemento vigilado.

4.4.5. Base de datos relacional

La implementación de la base de datos relacional se ha realizado mediante dos scripts en lenguaje SQL, un primer script para la creación de la base de datos que se almacena en el archivo *script.sql* y un segundo script para introducir los registros mínimos y un primer usuario con el rol de administrador para que el sistema funcione que se almacena en el archivo *insert.sql*. El contenido de ambos archivos figura en el "Anexo A7. Implementación de base de datos".

4.4.6. Aplicación para smartphone Android

4.4.6.1. Fuentes de terceros utilizadas

Para la implementación del diseño de la app se han utilizado las librerías **firebase-auth** versión 21.0.2 y **firebase-messaging** versión 23.0.0 elaboradas ambas por Google (2022), **lipeRMI** versión 0.4.2 (Santos Andrade, F., 2006) para el uso del servicio RMI y **PhotoView** versión 1.2.4 (Baseflow, 2022) para la visualización de imágenes permitiendo hacer zoom sobre ellas.

También se ha reutilizado y adaptado código propuesto por Google para la utilización de sus servicios Firebase Authentication y Firebase Cloud Messaging.

Estas fuentes se encuentran bajo las licencias GNU General Public License versión 2.1 (lipeRMI) y Apache 2.0 (librearías y código propuesto por Google y PhotoView).

Todo el código se encuentra disponible en el repositorio home-alarm de github.com bajo la licencia GNU General Public License versión 3, por lo que no se genera ningún conflicto derivado de la reutilización y distribución de las librerías y código.

4.4.6.2. Validación de usuario

En el código de la app se ha implementado la verificación de email en todas las actividades a través del método onCreate tal y como se puede ver en el Texto 5, de tal modo que la actividad en curso (<ActivityName>) se cierra y se abre la actividad WelcomeActivity.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mAuth = FirebaseAuth.getInstance();
    FirebaseUser currentUser = mAuth.getCurrentUser();
    if (currentUser == null || (currentUser != null && !currentUser.isEmailVerified())) {
        Intent intent = new Intent(<ActivityName>.this, WelcomeActivity.class);
        startActivity(intent);
        <ActivityName>.this.finish();
    }
}
```

Texto 5: Sobrescritura del método onCreate en todas las actividades de la app

4.4.6.3. Recepción de datos y notificaciones

La recepción de notificaciones del servicio Firebase Cloud Messaging de Google se realiza a través de la clase MyFirebaseMessagingService la cual se da de alta como servicio en el fichero AndroidManifest.xml como ilustra la Figura 20.

```
<service
    android:name="firebase.MyFirebaseMessagingService"
    android:exported="false">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>
</service>
```

Figura 20: Alta del servicio de atención a los mensajes de Firebase Cloud Messaging

La clase MyFirebaseMessagingService extiende la clase FirebaseMessagingService de la librearía firebase-messaging y sobreescribe la función onMessageReceived, que se puede ver en el “Anexo A8. Manejo de mensajes recibidos de Firebase”, la cual es llamada cuando la aplicación recibe un mensaje. Por un lado maneja el contenido de datos (duplas entrada-valor) difundiendo un mensaje a todas las actividades de la aplicación (local broadcast) con la información recibida que se encapsula en un objeto Intent, que podrá ser un cambio de estado de alarma, un código numérico tecleado en el control de acceso o una etiqueta RFID leída por el control de acceso. Finalmente verifica si hay carga de datos de

notificación y, si la hay, lanza una notificación llamando a una función implementada en la misma clase.

Los datos difundidos a todas las actividades de la aplicación serán atendidos solamente si la actividad ha activado el mecanismo receptor de este tipo de llamadas usando un objeto BroadcastReceiver. La actividad MainMenuActivity activa la atención de datos con contenido de cambios de estado, que se identifican con la cadena “ALARM_STATE”. Para filtrar los mensajes de difusión con el contenido de cambios de estado se utiliza un filtro implementado mediante un IntentFilter. El extracto de código que implementa todo lo explicado en este párrafo se puede ver con detalle en el “Anexo A9. Recepción de mensajes broadcast”.

4.4.6.4. Interfaz de usuario

La interfaz de ususario implementada se pude ver en el “Anexo A10. Interfaz usuario”.

4.5. Pruebas

El funcionamiento del sistema se valida mediante la realización de una serie de pruebas establecidas en base a los requisistos y casos de uso establecidos en epígrafes previos. Estas pruebas se han formalizado a través de casos de prueba que figuran en el “Anexo A11. Casos de prueba”. En el “Anexo A12. Relación entre casos de prueba y casos de uso” se muestra la correspondencia entre los casos de prueba y los casos de uso.

CAPÍTULO 5: EVALUACIÓN

[ToDo – Redactar epígrafe] La realización exitosa de cada uno de los casos de prueba enumerados en el epígrafe anterior asegura el funcionamiento deseado del sistema. Además, se han realizado una serie de verificaciones sobre la disponibilidad de RAM de la placa Arduino y de los tiempos consumidos por cada ciclo del proceso polling (loop, en inglés)…

Promedios de tiempos: (microsegundos)

- Duración de cada loop en estado Desarmado: 53,5
- Duración de cada loop en estado Armado
  - Con 0 sensores habilitados: 61
  - Con 1 sensor habilitado: 67,8
  - Con 2 sensores habilitados: 74,5
  - Con 3 sensores habilitados: 81,4
  - Con 4 sensores habilitados: 88,1
  - Con 5 sensores habilitados: 94,9
  - Con 6 sensores habilitados: 101,7
  - Con 7 sensores habilitados: 108,6
  - Con 8 sensores habilitados: 115,4
  - Con 9 sensores habilitados: 122,2
  - Con 10 sensores habilitados: 128,9
  - Con 11 sensores habilitados: 135,4
- Duración de cada loop en estado Disparado: 47,7

CAPÍTULO 6: CONCLUSIONES

[ToDo – Redactar epígrafe]

BIBLIOGRAFÍA

Agustina, J. R., & Reales, F. (2013). En la mente de un asaltante de viviendas: Estudio cualitativo de una muestra de autores de robo en casa habitada. *Revista Española De Investigación Criminológica, 11*, 1–30. https://doi.org/10.46381/reic.v11i0.73

Alexandres, S., Rodríguez-Morcillo, C., & Muñoz Frías, J. D. (enero, 2006). RFID: La tecnología de identificación por radiofrecuencia. *Anales de Mecánica y Electricidad*. *Vol. LXXXIII*, núm. I, pp. 47-52. https://www.iit.comillas.edu/publicacion/revista/es/45/RFID:_La_tecnolog%C3%ADa_de_identificaci%C3%B3n_por_radiofrecuencia

Álvarez Bayona, T. (2008). *Aspectos ergonómicos del ruido: evaluación*. Centro Nacional de Nuevas Tecnologías. Instituto Nacional de Seguridad e Higiene en el Trabajo. https://www.insst.es/documents/94886/509319/DTE-Aspectos+Ergonomicos+RUIDO+y+VIBRACIONES.pdf/f19b4be7-4f7d-4f11-9d12-b0507638290f

Álvarez Villavicencio, C. J. (marzo, 1991). *Control de acceso por tarjeta magnética y teclado* [Trabajo fin de carrera, Escuela Politécnica Nacional de Quito]. http://bibdigital.epn.edu.ec/handle/15000/10642

Bain, M. (2009). Comentarios breves sobre la GNU General Public License v3. *IDP: revista de Internet, derecho y política*, núm. 8, pp. 14-24. Universitat Oberta de Catalunya. https://dialnet.unirioja.es/servlet/articulo?codigo=3040674

Blanchon, B. (2022). ArduinoJson (Versión 6.19.4) [Biblioteca]. https://arduinojson.org/

Bogen, A. E., & Wollan, V. (1999). *AVR Enhanced RISC Microcontrollers*. ATMEL Development Center. http://ebook.pldworld.com/_Semiconductors/Atmel/Databook%20CDROM/Atmel/acrobat/riscmcu.pdf

Caballero, M. A., Ramos, L., & Saltijeral, M. T. (2000). El trastorno por estrés postraumático y otras reacciones en las víctimas del robo a casa. *Salud Mental (Mexico City, Mexico)*, 23(1), 8-17. https://www.redalyc.org/articulo.oa?id=58212302

Chatterjee, N., Chakraborty, S., Decosta, A., & Nath, A. (2018). Real-time communication application based on android using Google firebase. *Int. J. Adv. Res. Comput. Sci. Manag. Stud*, *6*(4).

Choque Zapana, M. A. (2018). *Sistema de alimentación ininterrumpida off-line para cargas críticas.* [Trabajo fin de carrera, Universidad Mayor de San Andrés, Ciudad de La Paz]. http://repositorio.umsa.bo/xmlui/handle/123456789/21997

Comunidad de Madrid. Decreto 219/2021, de 22 de septiembre, del Consejo de Gobierno, por el que se establecen las fiestas laborales para el año 2022 en la Comunidad de Madrid. Boletín Oficial de la Comunidad de Madrid, núm. 227, de 23 de septiembre, pp. 13-15. https://www.bocm.es/boletin-completo/BOCM-20210923/227

España. Ley 5/2014, de 4 de abril, de Seguridad Privada. *Boletín Oficial del Estado*, núm. 83, de 5 de abril de 2014, pp. 28975-29024. https://www.boe.es/eli/es/l/2014/04/04/5/con

España. Ley Orgánica 1/2015, de 30 de marzo, por la que se modifica la Ley Orgánica 10/1995, de 23 de noviembre, del Código Penal. Boletín Oficial del Estado, núm. 77, pp. 27061-27176. https://www.boe.es/eli/es/lo/2015/03/30/1

España. Ley Orgánica 10/1995, de 23 de noviembre, del Código Penal. *Boletín Oficial del Estado*, núm. 281, de 24 de noviembre, pp. 33987-34058. https://www.boe.es/eli/es/lo/1995/11/23/10

España. Ley Orgánica 15/2003, de 25 de noviembre, por la que se modifica la Ley Orgánica 10/1995, de 23 de noviembre, del Código Penal. *Boletín Oficial del Estado*, núm. 283, pp. 41842-41875. https://www.boe.es/eli/es/lo/2003/11/25/15

España. Orden INT/316/2011, de 1 de febrero, sobre funcionamiento de los sistemas de alarma en el ámbito de la seguridad privada. *Boletín Oficial del Estado*, núm. 42, 18 de febrero de 2011, pp. 18315-18332. https://www.boe.es/eli/es/o/2011/02/01/int316

Fang, Y. (2012). JSON-Simple (Versión 1.1.1) [Biblioteca]. https://github.com/fangyidong/json-simple

Farpointe Data, Inc. (19 de julio de 1999). *Pyramid Wiegand Data Format*. Farpointe Data, Inc. https://vkmodule.com.ua/pdf/Wiegand%20Format.pdf

Farpointe Data, Inc. (13 de febrero de 2020). *Card reader + keypad models P-620, P-640, Delta6.2 and Delta6.4 – Wiegand output*. Farpointe Data, Inc. https://www.farpointedata.com/downloads/qsg_and_ref/KeypadMode_REF.pdf

Fuertes Araque, J. (febrero, 2007). *Detector de movimiento por infrarrojos*. [Trabajo fin de carrera, Universitat Politècnica de Catalunya]. http://hdl.handle.net/2117/82100

Free Software Foundation. (29 de junio de 2007). *GNU General Public Licence*. http://www.fsf.org/licensing/licenses/gpl.html

Halfacree, G. (14 de marzo de 2018). *Benchmarking the Raspberry Pi 3 B+*. Medium. Consultado el 27 de marzo de 2022, de https://medium.com/@ghalfacree/benchmarking-the-raspberry-pi-3-b-plus-44122cf3d806.

Halfacree, G. (24 de junio de 2019). *Benchmarking the Raspberry Pi 4*. Medium. Consultado el 27 de marzo de 2022, de https://medium.com/@ghalfacree/benchmarking-the-raspberry-pi-4-73e5afbcd54b.

Halfacree, G. (2020). *Raspberry Pi 4 Firmware Updates Tested: A Deep Dive Into Thermal Performance and Optimization*. Hackster.io. Consultado el 27 de marzo de 2022, de https://www.hackster.io/news/raspberry-pi-4-firmware-updates-tested-a-deep-dive-into-thermal-performance-and-optimization-2f22c78e7089.

Hedgecock, W. (2022). JserialComm (Versión 2.9.1) [Biblioteca]. https://fazecast.github.io/jSerialComm/

HID Global Corporation. (12 de octubre de 2005). *Understanding Card Data Formats*. Technology Basics White Paper. HID Global Corporation (ASSA ABLOY). https://www.hidglobal.com/sites/default/files/hid-understanding_card_data_formats-wp-en.pdf

Hidalgo Garcés, R. (1980). *Detector sónico de movimiento* [Trabajo fin de carrera, Escuela Politécnica Nacional de Quito]. http://bibdigital.epn.edu.ec/handle/15000/10661

Instituto de Estadística de la Comunidad de Madrid. (25 de enero de 2020). *Estimación del parque de viviendas*. Ministerio de Transportes, Movilidad y Agenda de la Comunidad de Madrid. http://www.madrid.org/iestadis/fijas/basicos/cifras10_10.htm

Liew, J.P. (2020). Wiegand Protocol Library for Arduino (Versión 1.0.20) [Biblioteca]. https://github.com/monkeyboard/Wiegand-Protocol-Library-for-Arduino.git

Lizcano Casas, D. (2015). Sistemas distribuidos. Centro de Estudios Financieros.

López Mora, S. (17 de mayo de 2020). *Firebase: qué es, para qué sirve, funcionalidades y ventajas*. Digital55. https://www.digital55.com/desarrollo-tecnologia/que-es-firebase-funcionalidades-ventajas-conclusiones/

Lorenzo Bragado, C. (2013). *Bases de datos*. Centro de Estudios Financieros.

Mateos Costilla, D., & Reaño Montoro, S. (2008). Streaming de Audio/Video. Protocolo RTSP. Enginy revista de l'Escola Politècnica Superior de la Universidad de les Illes Balears, pp. 15-18. https://dialnet.unirioja.es/servlet/articulo?codigo=6399619

Microchip Technology. (6 de mayo de 2020). *ATmega640/1280/1281/2560/2561 - Complete Datasheet*. Microchip Technology Inc. https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega640-1280-1281-2560-2561-Datasheet-DS40002211A.pdf

Ministerio del Interior. (30 de septiembre de 2021). *Anuario Estadístico del Ministerio del Interior*. Secretaría General Técnica del Ministerio del Interior. http://www.interior.gob.es/web/archivos-y-documentacion/anuario-estadistico-de-2020

Mokar, M., Fageeri, S., & Fattoh, S. (2019). Using Firebase Cloud Messaging to Control Mobile Applications. *2019 International Conference On Computer, Control, Electrical, And Electronics Engineering (ICCCEEE)*. https://doi.org/10.1109/iccceee46830.2019.9071008

Oliver Gil, J. S. (2018). *El protocolo RTSP* [Webinar]. http://hdl.handle.net/10251/105099

ONVIF. (septiembre, 2018). *Profile T specification. Version 1.0*. ONVIF. Disponible en: https://www.onvif.org/wp-content/uploads/2018/09/ONVIF_Profile_T_Specification_v1-0.pdf

ONVIF. (mayo, 2011). *Application programmer's guide. Version 1.0.* ONVIF. http://www.onvif.org/wp-content/uploads/2016/12/ONVIF_WG-APG-Application_Programmers_Guide-1.pdf

Oracle Corporation. (2022). Mysql Connector Java (Versión 8.0.28) [Biblioteca]. https://dev.mysql.com/downloads/connector/j/

Orellana Peró, C. (mayo, 2020). Aplicación para el descubrimiento e integración de datos con protocolo ONVIF [Trabajo fin de carrera, Universidad Autónoma de Madrid]. http://hdl.handle.net/10486/693401

Ortigosa López, R., & Vázquez Ramiro, L. (1 de marzo de 2016). *Ingeniería del software*. Centro de Estudios Financieros.

Ricketts, M. (enero, 2022). Deafening siren. *Professional Safety, 67(1)*, 36-39.

Salazar Sanz, R. (15 de enero de 2020). *Guía básica de seguridad frente al robo en viviendas unifamiliares*. [Trabajo fin de carrera, Universidad Politécnica de Madrid]. https://oa.upm.es/58022/

Santos Andrade, F. (2006). LipeRMI (Versión 0.4.2) [Biblioteca]. http://lipermi.sourceforge.net/

Security Industry Association. (17 de octubre de 1996). *Access control standard protocol for the 26-bit Wiegand reader interface*. Security Industry Association.

Schulzrinne, H., Rao, A., Lanphier, R., Westerlund, M., & Stiemerling, M. (diciembre, 2016). *Real-Time Streaming Protocol Versoin 2.0*. Request for Comments 7826. Internet Engineering Task Force (IETF). https://datatracker.ietf.org/doc/html/rfc7826

ANEXOS

Anexo A1. Casos de uso

CU1. Identificarse	
Descripción	El usuario introduce su dirección de correo electrónico (email) y contraseña que es validado.
Actores	Usuario
Objetivo	Acceder a las funciones de uso del sistema que proporciona la app.
Precondiciones	**1.** El email del usuario ha debido ser dado de alta previamente por un administrador (CU7)
Flujo básico	**1.** El usuario rellena un formulario con su email y contraseña **2.** El usuario pulsa el botón enviar **3.** La app envía la información a Firebase (Google) **4.** Firebase valida al usuario **5.** La app envía el email al servidor de la alarma **6.** El servidor valida el email
Excepciones	**1.** Firebase no valida al usuario **2.** El servidor no valida al usuario En ambas excepciones se muestra al usuario un mensaje avisando a éste de que los datos no son válidos.
Postcondición	El usuario debe comprobar los datos introducidos y volver a enviar una petición de entrada

CU2. Armar	
Descripción	El sistema inicia el estado de vigilancia
Actores	Cualquier persona, Usuario (app)
Objetivo	Poner el sistema de alarma en estado Armado, en dicho estado el sistema vigila constantemente los distintos sensores.
Precondiciones	**1.** El estado del sistema debe ser Desarmado **2.** Si se pretende armar el sistema a través de la app, el usuario ha debido de identificarse (CU1)
Flujo básico	<u>Introduciendo el código numérico</u> **1.** El usuario introduce el código numérico de armado **2.** El sistema pasa a estado Prearmado durante el cual el usuario debe abandonar la vivienda en el lapso de tiempo de 20 segundos **3.** El sistema pasa a estado Armado <u>A través de la app</u> **1.** El usuario toca en la opción de armar el sistema **2.** El sistema pasa a estado Armado

CU3. Desarmar	
Descripción	El sistema pasa al estado Desarmado
Actores	Usuario
Objetivo	Poner el sistema de alarma en estado Desarmado.
Precondiciones	**1.** El administrador ha debido asociar al usuario un código y/o una etiqueta RFID (CU7) **2.** Si se pretende armar el sistema a través de la app, el usuario ha debido de identificarse (CU1)
Flujo básico	Introduciendo el código numérico **1.** El usuario introduce el código numérico de desarmado **2.** El código es validado y el sistema pasa a estado Desarmado Aproximando una etiqueta RFID **1.** El usuario aproxima una etiqueta RFID **2.** La etiqueta es validada y el sistema pasa a estado Desarmado A través de la app **1.** El usuario toca en la opción de desarmar el sistema **2.** El sistema pasa a estado Desarmado
Excepciones	**1.** El código introducido no se reconoce como válido **2.** La etiqueta RFID aproximada no se reconoce como válida
Postcondición	El usuario debe introducir nuevamente un código correcto o aproximar una etiqueta RFID válida.

CU4. Ver historial	
Descripción	La app muestra los últimos registros del sistemas
Actores	Usuario
Objetivo	Visualizar el historial de los últimos eventos registrados por el sistema.
Precondiciones	**1.** El usuario ha debido de identificarse (CU1)
Flujo básico	**1.** El usuario toca la opción de historial en la app **2.** La app muestra las últimas enésimas entradas registradas

CU5. Ver imágenes	
Descripción	La app muestra imágenes captadas por una cámara IP
Actores	Usuario
Objetivo	Visualizar una o más imágenes capturadas por una cámara IP dada de alta en el sistema de seguridad.
Precondiciones	**1.** El usuario ha debido de identificarse (CU1) **2.** La cámara ha sido dada de alta por el administrador (CU9)
Flujo básico	**1.** El usuario toca la opción de cámaras y navega (cámara/evento/imágenes) hacia la o las imágenes que quiere visualizar. **2.** El usuario toca sobre la imagen que quiere visualizar **3.** La app muestra la imagen seleccionada

CU6. Capturar imágenes	
Descripción	El sistema inicia una ráfaga de capturas de imágenes a petición del usuario
Actores	Usuario
Objetivo	Capturar una ráfaga de imágenes de una cámara determinada
Precondiciones	**1.** El usuario ha debido de identificarse (CU1) **2.** La cámara ha sido dada de alta por el administrador (CU9)
Flujo básico	**1.** El usuario toca la opción de cámaras y selecciona la cámara con la que quiere que se capturen las imágenes. **2.** El usuario toca la opción de capturar imágenes **3.** El sistema captura una serie de imágenes sobre la cámara seleccionada. La serie de imágenes consistirá en una imagen inicial y dos imágenes como mínimo posteriores a la primera en los siguientes 5 segundos.

CU7. Gestionar usuarios	
Descripción	El administrador dará de alta a los usuarios del sistema, configurará su perfil (alias, código numérico asociado y etiqueta RFID asociada) y dará de baja a los usuarios necesarios.
Actores	Administrador
Objetivo	Manejar el registro de usuarios del sistema
Precondiciones	**1.** El administrador ha debido de identificarse (CU1)
Flujo básico	<u>Alta de usuario</u> **1.** Tocará el botón “añadir usuario” dentro del apartado de ajustes / usuarios **2.** Rellenará un formulario con el email y un alias asociado al usuario **3.** Tocará el botón de aceptar y el sistema registrará el nuevo usuario <u>Asociación de código numérico para desarme</u> **1.** Entrará en el usuario correspondiente dentro del apartado de ajustes / usuarios **2.** Tocará el botón de “modificar código” **3.** Introducirá en el teclado del control de acceso el código deseado **4.** El sistema asociará el código al usuario y la app mostrará el código leído <u>Asociación de etiqueta RFID para desarme</u> **1.** Entrará en el usuario correspondiente dentro del apartado de ajustes / usuarios **2.** Tocará el botón de “modificar RFID” **3.** Aproximará la etiqueta RFID al control de acceso **4.** El sistema asociará la etiqueta RFID al usuario y la app mostrará el número de serie de la etiqueta leída <u>Otorgar o quitar rol de administrador</u> **1.** Entrará en el usuario correspondiente dentro del apartado de ajustes / usuarios **2.** Tocará el botón de “es administrador” **3.** El sistema dará o quitará el rol de administrador al usuario <u>Eliminar usuario</u> **1.** Entrará en el usuario correspondiente dentro del apartado de ajustes / usuarios **2.** Dejará pulsado el botón con el nombre del usuario **3.** La app mostrará un mensaje de confirmación de eliminación **4.** Si la respuesta es afirmativa, el sistema eliminará al usuario
Excepciones	En los flujos de quitar rol de administrador y eliminar usuario, si el usuario es el último administrador, el sistema mostrará el mensaje de que debe haber al menos un

	administrador en el sistema.
Postcondición	Se debe asegurar de no quitar el rol o eliminar a todos los administradores.

CU8. Gestionar sensores	
Descripción	El administrador habilitará los sensores con los que quiere trabajar, establecerá cuáles de ellos deben tener un disparo diferido y los asociará con una o más cámaras.
Actores	Administrador
Objetivo	Manejar la configuración de los sensores del sistema
Precondiciones	**1.** El administrador ha debido de identificarse (CU1) **2.** El administrador ha debido dar de alta a una o más cámaras (CU9)
Flujo básico	Habilitar o deshabilitar un sensor **1.** Seleccionará el sensor dentro del apartado de ajustes / sensores **2.** Tocará el botón “habilitado” **3.** El sistema habilitará o deshabilitará el sensor seleccionado Activar o desactivar el disparo diferido **1.** Seleccionará el sensor dentro del apartado de ajustes / sensores **2.** Tocará el botón “disparo retardado” **1.** El sistema activará o desactivará el disparo diferido para el sensor seleccionado Asociar con cámara(s) **1.** Seleccionará el sensor dentro del apartado de ajustes / sensores **2.** Tocará el botón “asociar con cámaras” **1.** Se activarán la cámara o cámaras con las que se quiere asocial el sensor Modificar alias **1.** Entrará en el usuario correspondiente dentro del apartado de ajustes / usuarios **2.** Tocará el botón de “cambiar alias” **3.** Se introducirá un nuevo alias y se validará
Excepciones	En el flujo de modificar alias, si el alias está siendo usado por otro sensor, se mostrará un mensaje de que ese nombre ya está siendo usado por otro sensor.
Postcondición	Se debe asignar un nombre de alias distinto y que no esté siendo usado por otro sensor.

CU9. Gestionar cámaras	
Descripción	El administrador dará de alta o baja en el sistema cámaras IP.
Actores	Administrador
Objetivo	Manejar la configuración de las cámaras IP del sistema
Precondiciones	**1.** El administrador ha debido de identificarse (CU1)
Flujo básico	Alta de una cámara **1.** Seleccionará el botón “añadir cámara” dentro del apartado de ajustes / cámaras **2.** Rellenar el formulario con el alias y la URI de la cámara IP. **3.** Aceptar y el sistema dará de alta la nueva cámara Eliminar una cámara **1.** Dejar presionado el nombre de la cámara a eliminar en el apartado de ajustes / cámaras **2.** El sistema mostrará un mensaje de confirmación para su eliminación **1.** Si la respuesta es afirmativa, el sistema eliminará la cámara del sistema y todas sus

	asociaciones con sensores.
Excepciones	En el flujo de alta de una cámara, si el alias está siendo usado por otra cámara, se mostrará un mensaje de que ese nombre ya está siendo usado por otra cámara.
Postcondición	Se debe asignar un nombre de alias distinto y que no esté siendo usado por otra cámara.

Anexo A2. Boceto y flujo de pantallas de la app

La Figura 21 muestra el proceso de identificación de usuario que contempla las funciones de registro y recordatorio de contraseña.

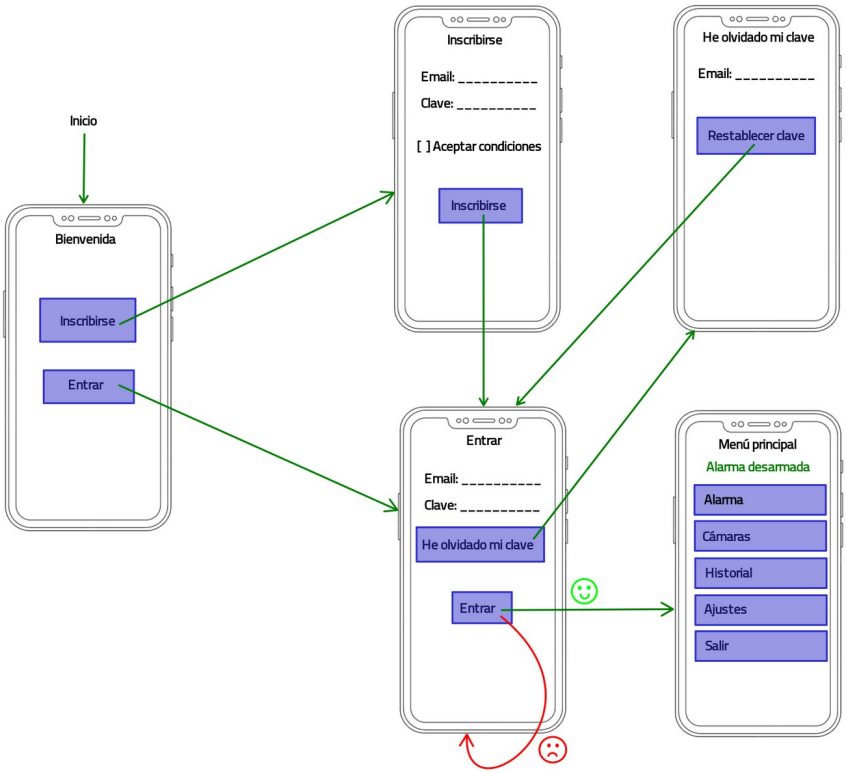


Figura 21: Boceto. Identificación de usuario y acceso al Menú principal

En la Figura 22, Figura 23, Figura 24 y Figura 25 se ilustra la navegación desde el Menú principal.

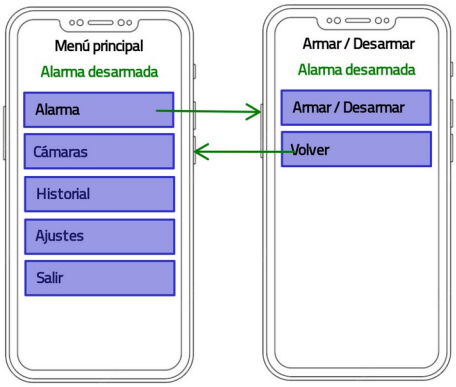


Figura 22: Boceto. Opción Alarma del Menú principal.

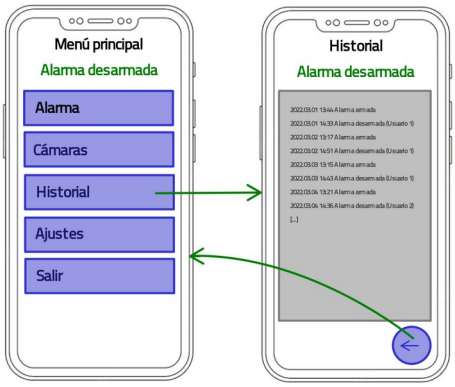


Figura 23: Boceto. Opción Historial del Menú principal.

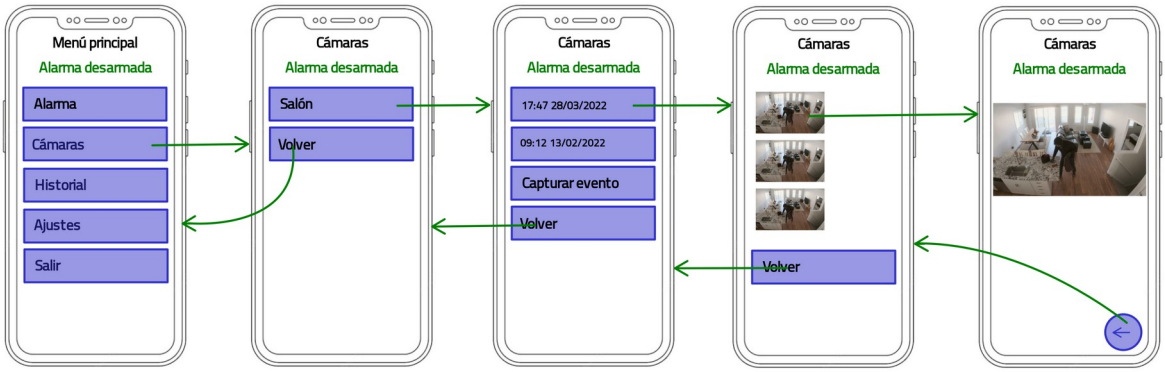


Figura 24: Boceto. Opción Cámaras del Menú principal.

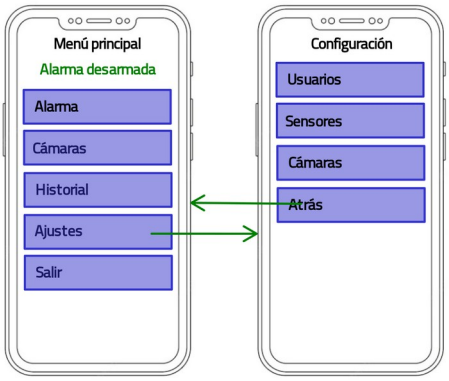


Figura 25: Boceto. Opción Configuración del Menú principal.

El apartado de Configuración, al que solamente tendrán acceso los usuarios con rol de administrador, se desarrolla desde la Figura 26 hasta la Figura 28.

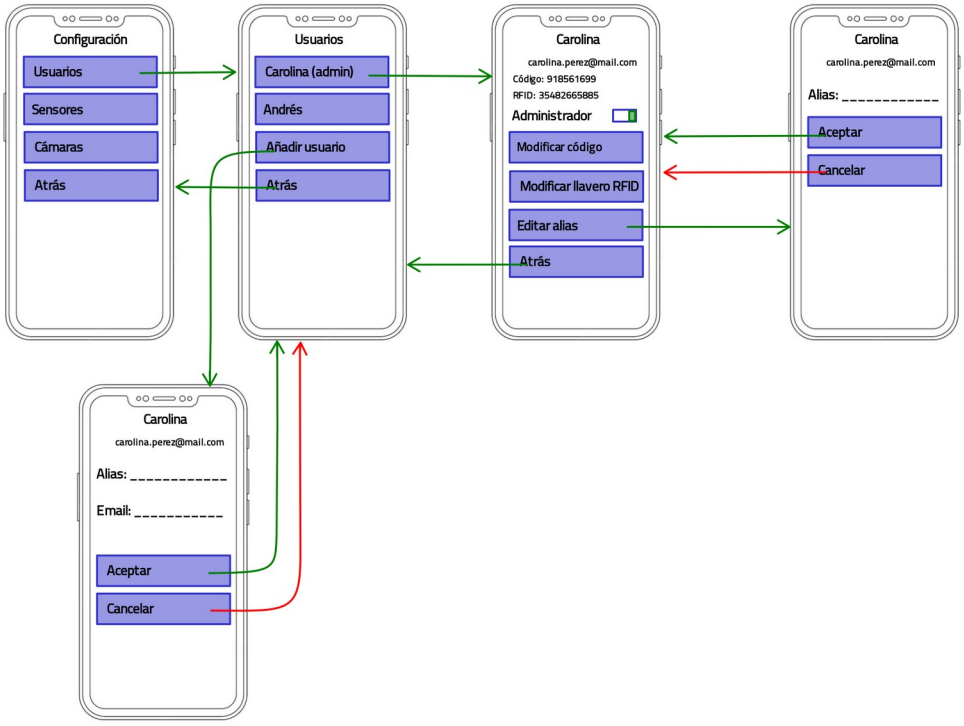


Figura 26: Boceto. Opción Usuarios del menú Configuración.

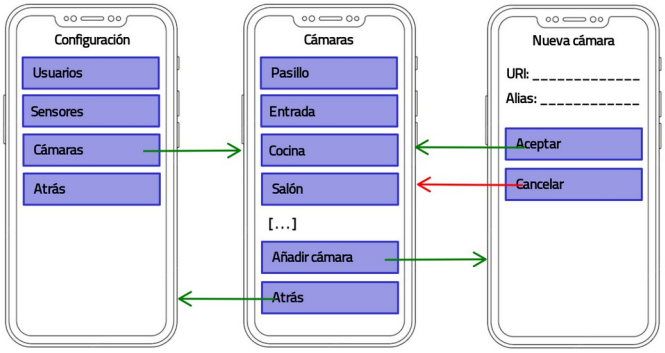


Figura 27: Boceto. Opción Cámaras del menú Configuración.

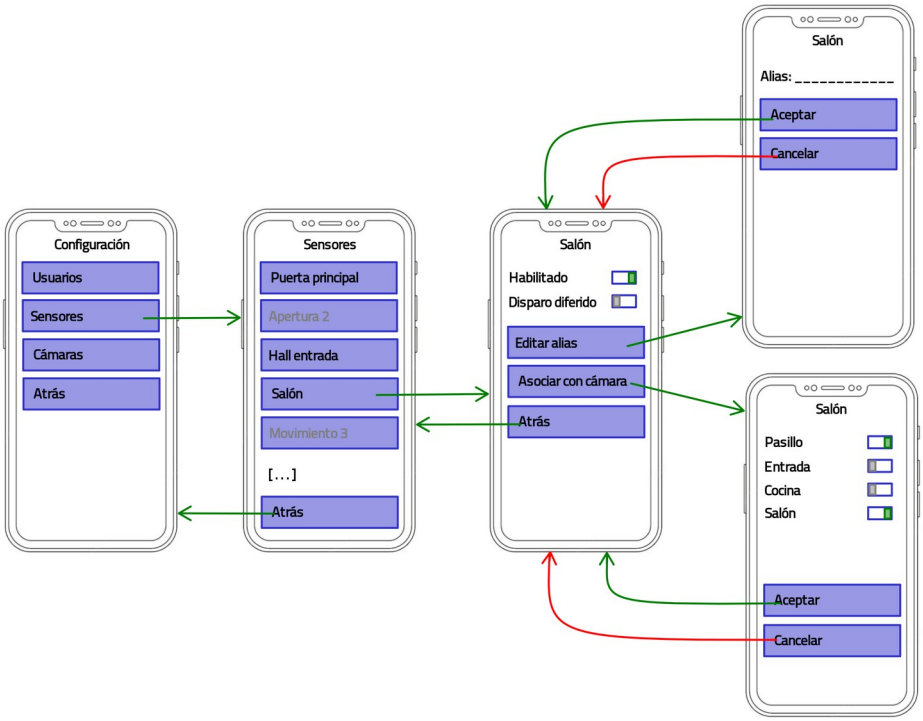


Figura 28: Boceto. Opción Sensores del menú Configuración.

Anexo A3. Mensajes centralita-servidor

Tabla 5: Mensajes enviados de la centralita al servidor

Tipo	Datos	Descripción
INICIO	<versión>	La centralita indica que finaliza la función setup() e inicia la función loop(). Contiene la versión.
ALARM_STATE	<id_estado>	Envía el código correspondiente al enum EstadoAlarma del estado actual del sistema
PREDISPARO	<id_sensor>	Notifica que el sensor *id_sensor* ha iniciado un predisparo
DISPARO	<id_sensor>	Notifica que el sensor *id_sensor* ha disparado el sistema de alarma. Si *id_sensor* es 999 significa que el sistema se ha disparado por una situación previa de predisparo.
CODIGO	<código>	Envía el código numérico leído por el control de acceso
TAG	<núm_serie>	Envía el número de serie de la etiqueta RFID leída por el control de acceso
FALLO_230	<estado>	Envía *true* si hay ausencia de alimentación 230V AC y *false* si todo está correcto
FALLO_BAT	<estado>	Envía *true* si la batería está baja y *false* si todo está correcto
SENSORS	<sensores>	Envía un array indicando el estado habilitado y disparo retardado de todos los sensores
RAM	<n_bytes>	Envía información de cuántos bytes libres de RAM quedan
LOOPS	<n_loops>	Envía la cantidad de ejecuciones de la función loop() en 10 segundos

Tabla 6: Mensajes enviados del servidor a la centralita

Tipo	Datos	Descripción
armar	<config.(1)>	La centralita carga la configuración de los sensores pasada y entra en estado Armado
prearmar	<config.(1)>	La centralita carga y guarda la configuración pasada y entra en estado Prearmado_inicial
desarmar	(sin datos)	La centralita borra la configuración guardada y pasa a estado Desarmado
getAlarmState	(sin datos)	La centralita envía su estado actual
getSensors	(sin datos)	La centralita envía la configuraciones de los sensores
getRam	(sin datos)	La centralita envía la cantidad de memoria RAM disponible
getLoops	(sin datos)	La centralita envía la cantidad de loops contados en 10 segundos

(1) El Texto 6 muestra un ejemplo de un mensaje enviado a la centralita para su armado o prearmado en el que se puede ver la configuración enviada.

```
{
  "tipo": "[armar | prearmar]",
  "datos": {
    "sensores": [
      {
        "id": 1,
        "habilitado": true,
        "retardado": true
      },
      {
        "id": 2,
        "habilitado": true,
        "retardado": false
      },
          [...]
      {
        "id": 11,
        "habilitado": false,
        "retardado": false
      }
    ],
    "codigos": [
      "918561699",
      "918561694",
       [...]
      "3587"
    ],
    "etiquetas": [
      1234567890,
      1234567891,
      [...]
      1234567892
    ]
  }
}
```

Texto 6: Ejemplo de mensaje JSON para armar o prearmar el sistema. Formateado con la herramienta online jsoneditoronline.org

Anexo A4. Servicio RMI

```
public void bind(final int port, final CallHandler callHandler, final IProtocolFilter
filter) throws IOException {
   enabled = true;
   serverSocket = new ServerSocket();
   serverSocket.setPerformancePreferences(1, 0, 2);
   serverSocket.bind(new InetSocketAddress(port));

   Thread bindThread = new Thread(new Runnable() {
      public void run() {
         while (enabled) {
            Socket acceptSocket = null;
            try {
               acceptSocket = serverSocket.accept(); // acción bloqueante

               final Socket clientSocket = acceptSocket;
               ConnectionHandler.createConnectionHandler(clientSocket,
                  callHandler,
                  filter,
                  new IConnectionHandlerListener() {
                  @Override
                  public void connectionClosed() {
                     for (IServerListener listener : listeners) {
                        listener.clientDisconnected(clientSocket);
                     }
                  }
               });
               for (IServerListener listener : listeners) {
                  listener.clientConnected(clientSocket);
               }
            } catch (IOException e) {
               e.printStackTrace();
            }
         }
      }
   }, String.format("alarma::ServicioRMI (%d)", port));
   bindThread.start();
}
```

Texto 7: Código de inicialización del servicio RMI

Anexo A5. Comunicación con Firebase Cloud Messaging

```
@Override
public void run() {
   HashMap<String, Object> mapNotificationPayload = new HashMap<>();
   if(titulo.length()>0 && mensaje.length()>0){
      mapNotificationPayload.put("title", titulo);
      mapNotificationPayload.put("body", mensaje);
      mapNotificationPayload.put("android_channel_id", androidChannel);
      mapNotificationPayload.put("click_action", "OPEN_ACTIVITY_1");
      mapNotificationPayload.put("image", uriImage);
   }
   if(!mapNotificationPayload.isEmpty() || !mapDataPayload.isEmpty()) {
      for(String token : tokens) {
         try{
            servidor.escribe("enviando mensaje Firebase a usuario...");
            URL url = new URL(urlFCM);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setUseCaches(false);
            conn.setDoInput(true);
            conn.setDoOutput(true);
            conn.setRequestMethod("POST");
            conn.setRequestProperty("Authorization", "key=" + claveServidor);
            conn.setRequestProperty("Content-Type", "application/json");
            JSONObject jsonMensaje = new JSONObject();
            jsonMensaje.put("to", token);
            jsonMensaje.put("priority", "high");
            jsonMensaje.put("time_to_live", 2419200);
            if(!mapNotificationPayload.isEmpty()){
               jsonMensaje.put("notification", new JSONObject(mapNotificationPayload));
            }
            if(!mapDataPayload.isEmpty()){
               jsonMensaje.put("data", new JSONObject(mapDataPayload));
            }
            try(OutputStreamWriter wr = new OutputStreamWriter(conn.getOutputStream())){
               wr.write(jsonMensaje.toJSONString());
               wr.flush();
            }
            servidor.escribe(conn.getResponseCode() + " - " + conn.getResponseMessage());
            conn.disconnect();
         } catch (IOException e) {
            servidor.escribe(e.toString());
         }
      }
   }
}
```

Texto 8: Envío de mensaje HTTP a Firebase

Anexo A6. Conexionado de elementos a la placa Arduino MEGA

Tabla 7: Conexionado de elementos a los pines de la placa Arduino MEGA

Pin	Descripción	Elemento conectado
0	Serial (Rx)	Tx (en lado servidor)
1	Serial (Tx)	Rx (en lado servidor)
2	INPUT (pull-up)	Sensor 1
3	INPUT (pull-up)	Sensor 2
4	INPUT (pull-up)	Sensor 3
5	INPUT (pull-up)	Sensor 4
6	INPUT (pull-up)	Sensor 5
7	INPUT (pull-up)	Sensor 6
8	INPUT (pull-up)	Sensor 7
9	INPUT (pull-up)	Sensor 8
10	INPUT (pull-up)	Sensor 9
11	INPUT (pull-up)	Sensor 10
12	INPUT (pull-up)	Sensor 11
18	INPUT (Interrupt)	Control de acceso (Wiegand, Data0)
19	INPUT (Interrupt)	Control de acceso (Wiegand, Data1)
22	INPUT (pull-up)	Fuente de alimentación (AC OK)
23	INPUT (pull-up)	Fuente de alimentación (Bat.)
40	OUTPUT	Sirena
45	OUTPUT	Control de acceso (LED R/V)
46	OUTPUT	Control de acceso (Buzzer)

Anexo A7. Implementación de base de datos

```
DROP SCHEMA IF EXISTS `alarma`;
CREATE SCHEMA IF NOT EXISTS `alarma` DEFAULT CHARACTER SET utf8;
USE `alarma`;

DROP TABLE IF EXISTS `alarma`.`Dispara`;
DROP TABLE IF EXISTS `alarma`.`Usuario`;
DROP TABLE IF EXISTS `alarma`.`Evento`;
DROP TABLE IF EXISTS `alarma`.`Sensor`;
DROP TABLE IF EXISTS `alarma`.`Camara`;

CREATE TABLE IF NOT EXISTS `Usuario`(
  `Email`             VARCHAR(100) NOT NULL PRIMARY KEY,
  `Alias`             VARCHAR(20) NOT NULL UNIQUE,
  `Es_administrador` BOOLEAN DEFAULT FALSE,
  `Token`             VARCHAR(255),
  `Tag_RFID`          INT,
  `Codigo`            VARCHAR(20)
);

CREATE TABLE IF NOT EXISTS `Evento`(
  `Id`                INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `Timestamp`         TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `Descripcion`       VARCHAR(255) NOT NULL
);

CREATE TABLE IF NOT EXISTS `Sensor`(
  `Id`                INT NOT NULL PRIMARY KEY,
  `Alias`             VARCHAR(20) NOT NULL UNIQUE,
  `Habilitado`        BOOLEAN DEFAULT FALSE,
  `Retardado`         BOOLEAN DEFAULT FALSE
);

CREATE TABLE IF NOT EXISTS `Camara`(
  `Alias`             VARCHAR(20) NOT NULL PRIMARY KEY,
  `Uri`               VARCHAR(255) NOT NULL
);

CREATE TABLE IF NOT EXISTS `Dispara`(
  `Id_sensor_dispara`       INT NOT NULL,
  `Alias_camara_disparada` VARCHAR(20) NOT NULL,
  PRIMARY KEY (`Id_sensor_dispara`,`Alias_camara_disparada`)
);

ALTER TABLE `alarma`.`Dispara`
  ADD FOREIGN KEY (`Id_sensor_dispara`) REFERENCES `alarma`.`Sensor` (`Id`),
  ADD FOREIGN KEY (`Alias_camara_disparada`) REFERENCES `alarma`.`Camara` (`Alias`) ON
UPDATE CASCADE;
```

Texto 9: Script contenido en fichero script.sql para la creación de la base de datos

```
USE `alarma` ;

INSERT INTO Usuario (Email,Alias,Es_administrador) VALUES
('fidelbreamontilla@gmail.com','Fidel',true);

INSERT INTO Sensor (Id,Alias) VALUES (1,'Sensor 1');
INSERT INTO Sensor (Id,Alias) VALUES (2,'Sensor 2');
INSERT INTO Sensor (Id,Alias) VALUES (3,'Sensor 3');
INSERT INTO Sensor (Id,Alias) VALUES (4,'Sensor 4');
INSERT INTO Sensor (Id,Alias) VALUES (5,'Sensor 5');
INSERT INTO Sensor (Id,Alias) VALUES (6,'Sensor 6');
INSERT INTO Sensor (Id,Alias) VALUES (7,'Sensor 7');
INSERT INTO Sensor (Id,Alias) VALUES (8,'Sensor 8');
INSERT INTO Sensor (Id,Alias) VALUES (9,'Sensor 9');
INSERT INTO Sensor (Id,Alias) VALUES (10,'Sensor 10');
INSERT INTO Sensor (Id,Alias) VALUES (11,'Sensor 11');
```

Texto 10: Script contenido en el archivo insert.sql para añadir registros iniciales

Anexo A8. Manejo de mensajes recibidos de Firebase

```
@Override
public void onMessageReceived(RemoteMessage remoteMessage) {
    if(remoteMessage != null) {
        System.out.println("onMessageReceived::From: " + remoteMessage.getFrom());
        Map<String, String> datos = remoteMessage.getData();
        if (remoteMessage.getData().size() > 0) {
            if(datos.get("alarm_state")!=null){
                Intent intent = new Intent("ALARM_STATE");
                intent.putExtra("alarm_state", datos.get("alarm_state"));
                LocalBroadcastManager.getInstance(getBaseContext()).sendBroadcast(intent);
            }
            if(datos.get("code")!=null){
                Intent intent = new Intent("CODE");
                intent.putExtra("code", datos.get("code"));
                LocalBroadcastManager.getInstance(getBaseContext()).sendBroadcast(intent);
            }
            if(datos.get("tag")!=null){
                Intent intent = new Intent("TAG");
                intent.putExtra("tag", datos.get("tag"));
                LocalBroadcastManager.getInstance(getBaseContext()).sendBroadcast(intent);
            }
        }
        if (remoteMessage.getNotification() != null) {
            sendNotification(remoteMessage.getNotification().getBody());
        }
    }
}
```

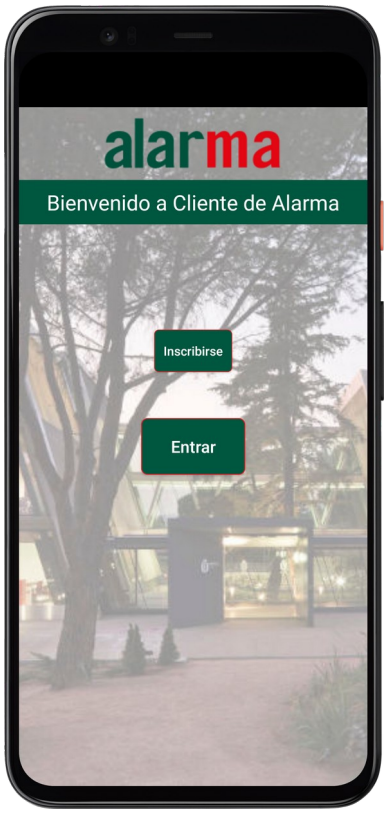
Texto 11: Función para el manejo de los mensajes recibidos de Firebase

Anexo A9. Recepción de mensajes broadcast

```
broadcastReceiver = new BroadcastReceiver() {
   @Override
   public void onReceive(Context context, Intent intent) {
      try {
         if (intent.getAction().equals("ALARM_STATE")) {
            switch(Integer.parseInt(intent.getExtras().get("alarm_state").toString())){
               case 0:
                  txtMessage.setText(getString(R.string.state_disarmed));
                  txtMessage.setTextColor(Color.WHITE);
                  txtMessage.setBackgroundColor(Color.rgb(0x00,0x57, 0x3F));
                  break;
               case 3:
                  txtMessage.setText(getString(R.string.state_armed));
                  txtMessage.setTextColor(Color.GREEN);
                  txtMessage.setBackgroundColor(Color.rgb(0x00,0x57, 0x3F));
                  break;
               case 6:
                  txtMessage.setText(getString(R.string.state_triggered));
                  txtMessage.setTextColor(Color.YELLOW);
                  txtMessage.setBackgroundColor(Color.RED);
                  break;
               default:
                  txtMessage.setText(getString(R.string.state_unknown));
                  txtMessage.setTextColor(Color.WHITE);
                  txtMessage.setBackgroundColor(Color.rgb(0x00,0x57, 0x3F));
            }
         }
      }catch(Exception e){
         e.printStackTrace();
      }
   }
};
IntentFilter filter = new IntentFilter("ALARM_STATE");
LocalBroadcastManager.getInstance(this).registerReceiver((broadcastReceiver), filter);
```

Texto 12: Recepción de mensajes broadcast

Anexo A10. Interfaz usuario



Anexo A11. Casos de prueba

Anexo A12. Relación entre casos de prueba y casos de uso

	CU1. Identificarse	CU2. Armar	CU3. Desarmar	CU4. Ver historial	CU5. Ver imágenes	CU6. Capturar imágenes	CU7. Gestionar usuarios	CU8. Gestionar sensores	CU9. Gestionar cámaras
CP01. Identificación erronea	x								
CP02. Identificación correcta	x								
CP03. Armar con código numérico		x							
CP04. Armar desde la app		x							
CP05. Desarmar con código numérico			x						
CP06. Desarmar con etiqueta RFID			x						
CP07. Desarmar desde la app			x						
CP08. Desarmado fallido. Salto de alarma			x						
CP09. Desarmar en estado Prearmado_inicial			x						
CP10. Desarmar en estado Prearmado_final			x						
CP11. Desarmar en estado Predisparado_inicial			x						
CP12. Desarmar en estado Predisparado_final			x						
CP13. Desarmar en estado Disparado			x						
CP14. Visualizar histórico				x					
CP15. Visualizar imágen capturada					x				
CP16. Capturar manualmente imágenes						x			
CP17. Capturar automáticamente imágenes						x			
CP18. Alta de usuario							x		
CP19. Asociación de código numérico a usuario							x		

	CU1. Identificarse	CU2. Armar	CU3. Desarmar	CU4. Ver historial	CU5. Ver imágenes	CU6. Capturar imágenes	CU7. Gestionar usuarios	CU8. Gestionar sensores	CU9. Gestionar cámaras
CP20. Asociación de etiqueta RFID a usuario							x		
CP21. Otorgar rol de administrador a usuario							x		
CP22. Quitar rol de administrador a usuario							x		
CP23. Eliminar usuario							x		
CP24. Habilitar sensor								x	
CP25. Deshabilitar sensor								x	
CP26. Activar disparo diferido de un sensor								x	
CP27. Desactivar disparo diferido de un sensor								x	
CP28. Asociar un sensor con cámaras								x	
CP29. Modificar alias de un sensor								x	
CP30. Añadir una cámara									x
CP31. Eliminar una cámara									x