



ESCUELA SUPERIOR DE INGENIERÍA

GRADO EN INGENIERÍA INFORMÁTICA

SISTEMA DE AVISOS PARA DISCAPACITADOS AUDITIVOS

Raúl Díez Sánchez

17 de enero de 2016



ESCUELA SUPERIOR DE INGENIERÍA

GRADO EN INGENIERÍA EN INFORMÁTICA

SISTEMA DE AVISOS PARA DISCAPACITADOS AUDITIVOS

- Departamento: Ingeniería Automática, Electrónica, Arquitectura y Redes de Computadores.
- Director del proyecto: Carlos Rodríguez Cordón
- Codirector del proyecto: Arturo Morgado Estévez
- Autor del proyecto: Raúl Díez Sánchez

Cádiz, 17 de enero de 2016

Fdo: Raúl Díez Sánchez

Agradecimientos

Quiero agradecer este trabajo a todas las personas que siempre han estado cerca, apoyándome y animándome a continuar en esta etapa de mi vida, gracias a ellos he conseguido cumplir uno de los grandes sueños de mi vida.

A mi familia, en especial a mis padres, por enseñarme que la vida es dura, y que hay que estudiar y trabajar para salir adelante en la vida. Por toda la confianza depositada en mí a lo largo de todos mis estudios.

A mis amigos de toda la vida, Iñaki, José, y María, que, desde el instituto en especial, me han apoyado en todo lo que he hecho, además de pasar muy buenos momentos que siempre me han servido para desconectar. También a todos los demás, los que han estado ahí, todos aportando su granito de arena en todo momento.

A todos mis compañeros, con los que de verdad he hecho amistad, en especial a Alejandro (a ambos), David, Ismael, Paco y Pepe, que tanta ayuda me han proporcionado y con los que tan buenos momentos he pasado a lo largo de mi etapa universitaria.

Por último, a todos los profesores que he tenido, en especial mis profesores del instituto y de la universidad, todos los que me han ayudado a ver que la informática era lo mío. A Javier, que desde el instituto me enseñó lo interesante que puede llegar a ser el mundo de los ordenadores. A Arturo y Carlos, mis tutores, quienes han sabido guiarme para que este trabajo llegase a buen puerto, y me han enseñado lo bonito que puede ser el tema que trato en él.

A todos, gracias. Se acabó.



SISTEMA DE AVISOS PARA DISCAPACITADOS AUDITIVOS

REF: 000001

ÍNDICE GENERAL

- **CLIENTE:** UNIVERSIDAD DE CÁDIZ (ESCUELA SUPERIOR DE INGENIERÍA)
AVENIDA DE LA UNIVERSIDAD DE CÁDIZ Nº 10, 11519 PUERTO REAL
956 48 32 00
DIRECCION.ESI@UCA.ES
- **AUTOR:** RAÚL DÍEZ SÁNCHEZ
INGENIERO INFORMÁTICO
77171812-G
RA.DIEZSANCHEZ@ALUM.UCA.ES

Cádiz, 17 de enero de 2016

Índice general

Índice de figuras

Índice de cuadros



SISTEMA DE AVISOS PARA DISCAPACITADOS AUDITIVOS

REF: 000001

MEMORIA

- **CLIENTE:** UNIVERSIDAD DE CÁDIZ (ESCUELA SUPERIOR DE INGENIERÍA)
AVENIDA DE LA UNIVERSIDAD DE CÁDIZ Nº 10, 11519 PUERTO REAL
956 48 32 00
DIRECCION.ESI@UCA.ES
- **AUTOR:** RAÚL DÍEZ SÁNCHEZ
INGENIERO INFORMÁTICO
77171812-G
RA.DIEZSANCHEZ@ALUM.UCA.ES

Cádiz, 17 de enero de 2016

Hoja de Identificación

- **Título del proyecto:** Sistema de avisos para discapacitados auditivos

- **Código del proyecto:** 000001

- **Datos del cliente:**
 - Universidad de Cádiz (Escuela Superior de Ingeniería)
 - Avenida de la Universidad de Cádiz N°10, 11519 Puerto Real
 - 956 48 32 00
 - direccion.esi@uca.es

- **Datos del suministrador:**
 - Raúl Díez Sánchez
 - Ingeniero Informático
 - Calle Sagasta N° 56 1ºA, 11002 Cádiz
 - 651 81 39 37
 - ra.diezsanchez@alum.uca.es

Firmado:

Cádiz, 17 de enero de 2016

Raúl Díez Sánchez

Cliente

1. Introducción

Cada día más, la tecnología nos muestra lo útil que puede llegar a ser para simplificar o ayudar en nuestro día a día. Un ejemplo de esto son la domótica y los proyectos AAL (o de vida asistida en la vivienda, por sus siglas en inglés) que surgen a raíz de la domotización.

La tecnología ayuda a solventar grandes necesidades, con soluciones cada vez mejores y más adaptadas. La accesibilidad es una de las grandes necesidades de la sociedad: permitir que un producto o servicio pueda ser utilizado por todos los posibles usuarios, incluyendo personas con diferentes capacidades o discapacidades.

Existen diversas discapacidades, y todas presentan un gran problema a la hora de que las personas que las padecen sean autónomas. Una de estas discapacidades es la auditiva. El simple hecho de estar en casa puede ser un gran problema, como lo es no saber cuándo llaman a la puerta.

2. Objetivo

El objetivo de este proyecto es crear un sistema para adaptar la vivienda de una persona o personas con discapacidad auditiva, de manera que puedan determinar en todo momento si alguien del exterior está intentando comunicarse con alguien en el interior de la vivienda.

Se pretende implantar un sistema de avisos visuales, que el discapacitado pueda ver, comprender y actuar conforme a la alerta.

3. Antecedentes

Actualmente existen varias soluciones a los problemas que pueda tener una persona con discapacidad auditiva, independientes del grado de discapacidad que padezca.

Una de las soluciones consiste en un aparato de bolsillo de avisos por señales de vibración y luces de colores [?], que recibe los avisos de distintos aparatos receptores de ruido que pueden obtener información del ruido generado por el timbre, el teléfono, el portero automático...

Esta solución requeriría que el discapacitado lleve consigo siempre un aparato en el bolsillo, lo cual puede ser un poco molesto. Además, tiene una limitación de número de alarmas a captar.

Otra posible solución es la instalación de ventanas en la mayoría de las paredes de la vivienda (paredes interiores), que permitiría ver las alertas visuales desde cualquier habitación. Este sistema necesitaría que previamente se modificasen los aparatos que emiten alertas sonoras para que emitan alertas visuales, además de reducir la intimidad de las personas que habiten en la

vivienda.

Una solución diferente a las anteriores es adaptar la instalación eléctrica de la vivienda [?], instalando tantas lámparas por habitación como alertas visuales se quieran mostrar. Estas lámparas mostrarían alertas visuales para el teléfono, el timbre, etc. La mayor desventaja aquí es el consumo de bombillas necesarias para una instalación completa, y el espacio que estas ocuparían.

4. Descripción de la Situación Actual

Hoy día, en cualquier vivienda existen demasiadas alertas sonoras como para controlarlas todas. Se pretende así por tanto tratar las alertas más importantes para que las personas con discapacidad puedan percibir los avisos más importantes.

4.1. Informe de diagnóstico

En este punto se tratan los distintos aspectos a tener en cuenta dentro de una vivienda estándar en cuanto a alertas se refiere. Este informe es la base de parte de los requisitos no funcionales.

4.1.1. Timbre de la puerta

Un caso es el timbre de la puerta, donde una persona con discapacidad auditiva es poco probable que sea capaz de percibir las alertas que genere.

4.1.2. Telefonillo (portero automático)

Otro caso es el del portero automático, del cual existen 2 tipos básicos: analógicos y digitales. Pese a que los telefonillos digitales son tecnológicamente superiores, se siguen utilizando telefonillos analógicos, debido a su bajo coste en comparación. Además, históricamente, el uso del telefonillo analógico es más extendido. Debido a estas razones, en este Proyecto se tendrá en cuenta el uso de telefonillo analógico, para abarcar un mayor rango de posibles instalaciones.

Entre los telefonillos analógicos, existen 2 tipos atendiendo al tipo de llamada que realizan:

- Llamada zumbador.
- Llamada electrónica.

La diferencia principal entre ambos reside en su funcionamiento y la forma de generar la alerta sonora [?].

- **Telefonillos analógicos de llamada zumbador:** En los telefonillos con llamada zumbador, una vez que se realiza la llamada, el grupo fónico de la placa exterior a la vivienda genera una señal alterna de 12 Vac que envía por el hilo de llamada. La señal alterna llega al zumbador situado en el teléfono, haciendo que el zumbador vibre y finalmente produzca el sonido de la llamada.
- **Telefonillos analógicos de llamada electrónica:** En los telefonillos con llamada electrónica, una vez que se realiza la llamada, el grupo fónico de la placa exterior a la vivienda genera una señal formada por diferentes frecuencias, y la envía por el hilo de llamada. La

señal llega al altavoz del auricular del teléfono, produciendo el sonido de la llamada. La llamada puede ser de varios tipos en función del número de frecuencias utilizadas.

Teniendo en cuenta que los de llamada zumbador envían una señal de 12 Vac, y los de llamada electrónica una señal de varias frecuencias, es más fácil trabajar con los del primer tipo, pudiendo darse soluciones electrónicas bastante sencillas. Además, son el tipo más extendido, y el que más probabilidades hay de encontrar en una vivienda. Se utilizará por tanto un telefonillo de este tipo para la implementación del sistema y la demostración de funcionamiento.

4.1.3. Bombillas

En cuanto a las bombillas existentes en una vivienda, las más empleadas son las bombillas de casquillo E27, también llamadas bombillas de rosca Edison. Por tanto, para abarcar un mayor rango de posibles instalaciones, este Proyecto se basará en el uso de bombillas de este tipo.

En caso de manipular estas bombillas, existen dos posibilidades: hacerlo mediante la propia instalación eléctrica, o manipularlas de manera inalámbrica.

5. Normas y Referencias

5.1. Disposiciones legales y Normas aplicadas

- **UNE 157801:2007** - Criterios generales para la elaboración de proyectos de sistemas de información.
- **UNE 157001:2014** - Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico.

5.2. Métodos, herramientas, métricas y prototipos

5.2.1. Herramientas software

Android Studio - Software para el desarrollo de aplicaciones Android proporcionado por Google, utilizado para la programación de la aplicación desarrollada en este Proyecto.

Eagle - Software de creación de circuitos electrónicos para su implementación en placas de circuito impreso. Utilizado para crear las PCB utilizadas en el Proyecto.

NinjaMock - Herramienta online para la creación de mockups o prototipos de diseño de aplicaciones, empleado en la realización del prototipado de la aplicación.

Python - Lenguaje de programación en el que se basa el sistema principal de este Proyecto, cada vez más extendido.

Sublime Text - Editor de texto especialmente diseñado para código, con multitud de complementos para auto-generar código rápidamente.

5.2.2. Herramientas mecánicas

Fresadora PCB LPKF ProtoMat S103 - Utilizada para obtener las placas de circuito impreso utilizadas en el Proyecto.

5.3. Otras referencias

5.3.1. Repositorio en GitHub

Tanto la documentación como el código fuente asociado descrito en este documento, están alojados en GitHub, disponibles en <https://github.com/raudiez/tfg>.

6. Definiciones y Abreviaturas

AAL Ambient Assisted Living (Vida Asistida en la Vivienda).

Arduino Plataforma electrónica de hardware libre de fácil uso, utilizada normalmente para el montaje de dispositivos multifuncionales.

Bluetooth Especificación industrial para WPAN que posibilita la transmisión de datos entre diferentes dispositivos con corto alcance.

Domótica Conjunto de sistemas capaces de automatizar una vivienda.

Ethernet Estándar de red local para computadores (IEEE 802.3).

GPIO General Purpose Input/Output (Entrada/Salida de Propósito General).

IDE Integrated Development Environment, o Ambiente de Desarrollo Integrado, es una herramienta software que proporciona servicios integrales en el desarrollo software.

IEEE Institute of Electrical and Electronics Engineers (Instituto de Ingeniería Eléctrica y Electrónica).

JSON JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos, muy simple y extendido, y representa la estructura de los datos en el propio texto.

LAN Local Area Network (Red de Área Local).

PAN Personal Area Network (Red de Área Personal).

PCB Printed Circuit Board (Placa de Circuito Impreso).

Piconet Red informática cuyos nodos se conectan utilizando Bluetooth. Puede constar de 2 a 7 dispositivos.

Raspberry Pi Computador embebido de bajo coste desarrollado en Reino Unido por la Raspberry Pi Foundation.

REST Representational State Transfer, es un estilo de arquitectura software para sistemas hipermedia distribuidos, y describe cualquier interfaz entre sistemas que utilice directamente métodos HTTP (GET, POST, PUT y DELETE) realizar operaciones con datos.

WEP Wired Equivalent Privacy (Privacidad Equivalente a Cableado).

WiFi WiFi (o Wi-Fi) es el nombre de una marca comercial de Wi-Fi Alliance, definido como cualquier producto basado en conexión WLAN que cumple con el estándar IEEE 802.11. Comúnmente se confunde con ese estándar, IEEE 802.11.

WLAN Wireless Local Area Network (Red de Área Local Inalámbrica).

WPA Wi-Fi Protected Access (Acceso Wi-Fi Protegido).

WPAN Wireless Personal Area Network (Red de Área Personal Inalámbrica).

7. Requisitos Iniciales

A partir de la encuesta realizada, disponible en el apartado ??, junto con información facilitada por el Cliente, se obtienen los requisitos iniciales del Proyecto:

- Se deben detectar las llamadas de timbre.
- Se deben detectar las llamadas de telefonillo.
- Alertar de los avisos mediante luces de distintos colores.
- Conectar smartphones Android al sistema de avisos.
- Notificar las alertas en smartphones Android.

Estos requisitos se exponen de una manera más detallada en el apartado ??, “Especificaciones del Sistema”.

8. Alcance

El alcance de este proyecto engloba los siguientes puntos:

- Desarrollo de la especificación de requisitos del sistema y su análisis.
- Diseño de la arquitectura general del sistema.
- Diseño de la interfaz de conexión entre los dispositivos generadores de alertas y los nodos informativos visuales.
- Elección de la tecnología de comunicación a emplear en el sistema.
- Implementación del software del sistema.
- Implementación del sistema con dos nodos informativos visuales como demostración.

9. Estudio de Alternativas y Viabilidad

Para la realización del Proyecto, se han tenido en cuenta diversas alternativas para los diferentes aspectos del Proyecto, ya sea la plataforma a utilizar para la recogida de datos de los generadores de alertas, la forma de generar las alertas visuales o la comunicación entre los diferentes elementos que engloben el sistema, entre otros.

Por ello se analizarán en distintos apartados las alternativas para cada caso.

9.1. Nodos informativos visuales

Teniendo claro a raíz del estudio de requisitos y de los resultados obtenidos en las encuestas realizadas, disponible en el apartado ??, los elementos principales que conformarán las alertas visuales serán luces RGB, a modo de bombillas de rosca E27 de acuerdo con los requisitos no funcionales NFR-01 y NFR-03, que puedan ser fácilmente implantadas en una vivienda real; y algún smartphone Android que conectar al sistema, de acuerdo con el requisito no funcional NFR-02, que recibirá las alertas en forma de notificación de texto.

Se analizaron por tanto las principales bombillas LED del mercado que fuesen controlables mediante alguna tecnología de red, para poder comunicar estas con el nodo central de recogida y envío de información. Las bombillas tenidas en cuenta fueron las que se muestran en la tabla ??.

Ya sea a través de ZigBee o WiFi, queda claro observando la tabla que las 3 opciones de bombillas disponibles requieren de un puente inalámbrico al que conectar, que será el controlador principal de las bombillas y su estado. En todo caso, estos puentes deben ir conectados a través de red local al nodo central que manejará las alertas, ya sea mediante WiFi o cable Ethernet.

Modelo	Color	Comunicación	Precio	Observaciones
Philips Hue [?]	RGB	<ul style="list-style-type: none"> ▪ Puente-bombillas: ZigBee ▪ Conexión a puente mediante Ethernet 	<ul style="list-style-type: none"> ▪ Starter Kit: 199.95 € [?] ▪ Bombilla E27: 52.99 € [?] 	El puente no se vende por separado, es necesario adquirir el Starter Kit.
Belkin WeMo [?]	Blanco	<ul style="list-style-type: none"> ▪ Puente-bombillas: ZigBee ▪ Conexión a puente mediante WiFi 	<ul style="list-style-type: none"> ▪ Starter Kit: 99.90 € [?] ▪ Bombilla E27: 29.99 € [?] 	El puente no se vende por separado, es necesario adquirir el Starter Kit.
LimitLessLED [?]	RGB	<ul style="list-style-type: none"> ▪ Puente-bombillas: WiFi ▪ Conexión a puente mediante WiFi 	<ul style="list-style-type: none"> ▪ Starter Kit: 62.45 € [?] ▪ Bombilla E27: 20.82 € [?] ▪ Puente: 17.65 € [?] 	Puente disponible por separado.

Cuadro 9.1: Comparativa de bombillas LED

9.2. Conexión nodo central-smartphone

Como se ha comprobado, en el sistema se debe contemplar el uso de un smartphone Android que reciba las mismas alertas a modo de texto, para que el usuario pueda recibir las alertas de varias formas, pudiendo no estar atento a las luces de la vivienda o pudiendo tenerlas apagadas.

Será necesario por tanto conectar de alguna forma este dispositivo al sistema. Existen en esencia 2 alternativas para ello:

- Conexión mediante red WiFi (IEEE 802.11).
- Conexión mediante Bluetooth.

Para la primera opción, teniendo en cuenta que las bombillas utilizadas también requerirán conexión de red, lo más apropiado será utilizar un router WiFi con capacidad para varias conexiones Ethernet. De este modo, el nodo central podrá conectarse al router WiFi ya sea mediante WiFi o cable Ethernet, y a su vez estarán conectados también al router el puente controlador de las bombillas, y el smartphone Android.

Para la segunda opción, será necesario que el nodo central cuente con algún dispositivo de comunicación Bluetooth, mediante el cual enviar las alertas al smartphone Android.

9.2.1. Bluetooth

El Bluetooth es un tipo de red WPAN de los más utilizados en el mercado. Cualquier smartphone Android cuenta hoy día con conexión Bluetooth. Aunque la distancia que cubre es relativamente pequeña (10 metros normalmente, hasta 100 metros utilizando repetidores), el objetivo de esta tecnología es sustituir los cables de interconexión entre dispositivos, objetivo que cumple porque la mayoría de estas conexiones no superan los 10 metros de distancia.

9.2.1.1. Especificaciones técnicas

Las especificaciones técnicas principales del estándar Bluetooth de interés para este proyecto se recogen en la tabla ??.

Banda de Frecuencia	2,4 GHz.
Velocidad de datos	Hasta 721 kbps por piconet.
Distancia máxima	10 m.

Cuadro 9.2: Especificaciones técnicas principales de interés del estándar Bluetooth.

9.2.1.2. Seguridad

Bluetooth define tres niveles de seguridad:

- **No seguro:** no se emplea ningún mecanismo de seguridad.
- **Seguridad a nivel de servicio:** permiso concedido por servicio (se puede acceder a un servicio pero no al resto).

- **Seguridad a nivel de enlace:** requiere de autenticación y autorización del dispositivo al que desea conectarse.

El último es el nivel más seguro y el más extendido en la actualidad.

9.2.1.3. Aplicaciones

Las características del estándar Bluetooth hacen de esta conexión adecuada para:

- Conexión a otras redes a través de otro dispositivo que funcione como punto de acceso.
- Conexión entre periféricos y dispositivos, sustituyendo el cable por un enlace inalámbrico.
- Conexión entre dispositivos para transferencia de datos y sincronización.

En resumen, una conexión Bluetooth es apta para cualquier conexión entre dispositivos que no requiera un gran alcance (10 metros).

9.2.2. WiFi (IEEE 802.11)

Aunque comúnmente conocido como WiFi (o Wi-Fi), el estándar IEEE 802.11 es un conjunto de especificaciones que implementan WLAN en las bandas de frecuencia 2.4 GHz, 3.6 GHz, 5 GHz y 60GHz. Inicialmente, Wi-Fi es el nombre de una marca comercial de la WECA (actualmente Wi-Fi Alliance), organización comercial que aprueba y certifica que los dispositivos de comunicación WLAN cumplen con el estándar IEEE 802.11. Actualmente, la misma Wi-Fi Alliance define “Wi-Fi” como cualquier producto basado en conexión WLAN que cumple con ese estándar IEEE 802.11. Aún así, es tan extendido el uso de la palabra “Wi-Fi” que resulta raro hablar del estándar. En este documento, cuando se emplee la palabra “Wi-Fi”, se referirá a una red WLAN que cumpla con el estándar IEEE 802.11.

La distancia que es capaz de cubrir es bastante amplia (si el dispositivo cumple con 802.11g u 802.11n), pudiendo llegar hasta los 100 metros con facilidad.

9.2.2.1. Especificaciones técnicas

Las especificaciones técnicas principales del estándar 802.11n (utilizado actualmente) de interés para este proyecto se recogen en la tabla ??.

Banda de Frecuencia	2,4 GHz y 5 GHz.
Velocidad de datos	Hasta 150 Mbps.
Distancia máxima	70 m en interior, 250 m en exterior.

Cuadro 9.3: Especificaciones técnicas principales de interés del estándar IEEE 802.11n.

9.2.2.2. Seguridad

Actualmente existen varias formas de securizar una red WiFi, entre las que destacan:

- **Sin seguridad:** Red abierta, sin necesidad de autenticar el dispositivo a conectar.

- **WEP:** Cifrado muy común en los inicios de esta tecnología, se compone de claves de entre 64 y 128 bits. Cada cierto tiempo, el dispositivo conectado envía parte de la clave al dispositivo que genera la conexión de red, para verificar que está conectado a la red. Esto supone una brecha importante de seguridad.
- **WPA:** Cifrado que mejora a su predecesor WEP, e incluye otras mejoras, generación dinámica de claves, y anticipa el entonces en desarrollo estándar IEEE 802.11i. Además palia los fallos de seguridad del cifrado WEP. Incrementa el tamaño de las claves y el número en uso e introduce un nuevo mensaje de control de integridad más seguro.
- **WPA2:** Mejora de WPA, que cambia el esquema de encriptado utilizado por AES, por lo que mejora nuevamente en nivel de seguridad.

9.2.2.3. Aplicaciones

Gracias a sus características, y sobre todo a su amplia distancia máxima que cubre, hacen de esta tecnología adecuada para:

- Conexión a otras redes a través de un dispositivo que funcione como punto de acceso.
- Conexión entre periféricos, cada vez más extendido, gracias a la inclusión de chips con conexión WiFi en los dispositivos más modernos.
- Conexión entre dispositivos para transferencia de datos, sincronización y creación de redes locales con otros fines.

En resumen, una conexión WiFi es apta para cualquier conexión entre dispositivos, con la posibilidad de abarcar una gran distancia (100 metros).

9.3. Plataforma para nodo central

El sistema deberá contar con un nodo central que maneje la información recibida, y genere las alertas necesarias, activando las alertas visuales mediante luces, y enviando las mismas alertas al smartphone Android. Este nodo central estará integrado en el sistema implementado en la vivienda, y no será visible ni accesible por el usuario final una vez instalado.

Por ello, se han tenido en cuenta dos alternativas: utilizar una placa Arduino, o usar una computadora Raspberry Pi.

9.3.1. Arduino

Arduino es una plataforma de hardware libre, basada en una placa con microcontrolador. En concreto, para este Proyecto se ha tenido en cuenta la placa Arduino Mega, al ser una de las placas Arduino con mayor capacidad de memoria flash programable.

La programación de estas placas es bastante sencilla, bastando con conectar la placa mediante su conector USB a un ordenador, y utilizar el entorno de desarrollo de Arduino (o cualquier editor que admita el uso de este entorno como plug-in) para escribir y cargar los programas en la placa.

9.3.1.1. Características técnicas

La Arduino Mega está basada en el microcontrolador ATmega1280, es una de las de mayor tamaño, sus pines digitales operan con una corriente de 5 V y sus características principales son las reflejadas en la tabla ?? [?].

Pines I/O digitales	54, de los cuales 15 proporcionan salida PWM.
Pines de entrada analógica	16.
Memoria flash	128 kB, de los cuales 4 kB son utilizados por el bootloader.
Alimentación	USB o conector jack, con voltaje máximo recomendado de entre 7 y 12 V.
Consumo	~ 50 mA, 0.3 W.

Cuadro 9.4: Características técnicas principales de interés de Arduino Mega.

Las placas Arduino son cada vez más utilizadas en entornos donde el funcionamiento del sistema será siempre el mismo, sin desaprovechar así un sistema operativo interno ni funcionalidades extra no necesarias.

Además, Arduino cuenta con una gran variedad de placas adicionales para dotar a sus placas Arduino de características o conexiones que no están disponibles por defecto.

9.3.1.2. Estudio de viabilidad de la alternativa

Como paso previo a la solución final, se estudió la posibilidad de utilizar Arduino como nodo central del sistema para la solución del Proyecto. Por ello, es necesario tener en cuenta en este apartado la elección de solución para cada alternativa anteriormente mencionada, expuesta en el apartado ??.

Es necesario por tanto acoplar una Shield propia de Arduino a la Arduino Mega, para dotarla de conexión a la red. Existen dos alternativas: utilizar una Shield WiFi, o utilizar una Shield Ethernet. Con la segunda opción se asegura una conexión estable (cable). Además, la Shield WiFi ha sido descatalogada como producto oficial de Arduino, y ya no se le da soporte ni a la placa ni a la documentación sobre esta. Por tanto se utilizará una Shield Ethernet para conectar la Arduino a la red, y con ello conectar con el smartphone Android y con el puente Hue de Phillips.

La Shield Ethernet utilizada monta un chip Ethernet nuevo (WIZnet W5500) para el que, a fecha de realización de este estudio, Arduino no da soporte a través de su entorno de desarrollo, por lo que es necesario utilizar una librería del propio fabricante del chip [?].

Para la comunicación con el puente Philips Hue, es necesario realizar peticiones HTTP mediante una interfaz RESTful [?] (peticiones GET, PUT, POST y DELETE), de acuerdo además con el requisito no funcional NFR-04. El puente además gestionará las peticiones y dará como resultado a estas, respuestas en formato JSON, por lo que será necesario gestionar estas cadenas formateadas para poder obtener los datos necesarios facilitados por la interfaz del puente. De acuerdo a la API de Philips Hue, las peticiones y respuestas se dan únicamente en formato JSON, por lo que se hace obligatorio utilizarlo (requisito no funcional NFR-05). Arduino no cuenta entre las librerías de su entorno con ninguna herramienta para el manejo de cadenas JSON (parser de

JSON), por lo que de entre todas las posibilidades existentes para el manejo de estas cadenas en lenguaje C/C++, se estudian dos alternativas posibles desarrolladas con licencia de software libre: utilizar la librería ArduinoJson desarrollada por Benoît Blanchon disponible en GitHub [?], o utilizar la librería aJson desarrollada por el equipo Interactive Matter disponible también en GitHub [?].

La primera librería, ArduinoJson, está más optimizada para su uso en placas Arduino, gestionando de mejor manera la corta memoria programable de la placa. La segunda, es una librería para manejo de cadenas JSON, desarrollada también para placas Arduino, pero sin optimizar tanto la memoria, sino centrándose en las funcionalidades. La segunda librería es más completa, pero la primera está más optimizada. Debido a la escasa memoria con la que cuenta Arduino, se decidió utilizar la librería ArduinoJson, para optimizar el uso de recursos de la placa.

Con Arduino además, al estar bastante limitado en lo que a memoria se refiere, se encuentran problemas al utilizar cadenas de caracteres en objetos de la clase String, los cuales consumen mucha memoria. Además, entre las implementaciones de la clase String con las que cuenta Arduino en su entorno de desarrollo, String es la más ligera de las implementaciones, contando además con WString o TextString dentro de las librerías del entorno. Esto hace que sea necesario utilizar variables de tipos más ligeros, como el tipo char, utilizando arrays dinámicos de este tipo.

Tras varias pruebas, se decidió además optar por usar una adaptación de la librería de software libre para el manejo de peticiones REST, RestClient [?], optimizando el uso de recursos de la librería, y cambiando el uso de variables de la clase String por arrays dinámicos de caracteres char. Esta librería modificada puede consultarse en el código asociado a este documento, disponible también en un repositorio en GitHub (apartado ??).

Para comprobar el uso de memoria que tenía el programa en la placa Arduino, se utilizó un sencillo código puesto a disposición en la web oficial de Arduino [?][?], el cual examina la memoria disponible restante de la placa.

```
1 #include <Ethernet.h>
2 #include <SPI.h>
3 #include "RestClient.h"
4 //https://github.com/bblanchon/ArduinoJson
5 #include "ArduinoJson.h"
6 //http://playground.arduino.cc/Code/AvailableMemory
7 #include "MemoryFree.h"
8 byte mac[]={ 0x90, 0xA2, 0xDA, 0x0F, 0xF9, 0x41 }; //MAC
9 byte ip[] = { 192, 168, 2, 146 }; //IP
10 const char* hueuser = "d3dede62b969c73f21574a17b7a8fb";
11
12 RestClient hue = RestClient("192.168.2.121");
13
14 StaticJsonBuffer<200> jsonBuffer;
15
16 void setup(){
17   Serial.print("Inicio - freeMemory(=)");
18   Serial.println(freeMemory());
19   Serial.begin(9600);
20   Ethernet.begin(mac,ip);
21   Serial.print("Tras conectar por ethernet - freeMemory(=)");
```

MEMORIA

```

22   Serial.println(freeMemory());
23 }
24
25 void loop() {
26   initHue();
27   Serial.print("freeMemory(=");
28   Serial.println(freeMemory());
29   int n = getNumHueLights();
30   Serial.println(n);
31   delay(5000);
32 }
33
34 void initHue() {
35   hue.setContentType("application/json");
36 }
37
38 char* getHueList() {
39   char* response;
40   char* aux = new char[13+strlen(hueuser)];
41   strcpy(aux, "/api/");
42   strcat(aux, hueuser);
43   strcat(aux, "/lights");
44   const char* path = aux;
45   int status = hue.get(path, response);
46   return response;
47 }
48
49 int getNumHueLights() {
50   char* json = getHueList();
51   Serial.print(json);
52   JsonObject& root = jsonBuffer.parseObject(json);
53   int num = 0;
54   for(JsonObject::iterator it=root.begin(); it!=root.end(); ++it) num++;
55
56   return num;
57 }

```

En el último programa de test realizado para Arduino, los resultados obtenidos seguían siendo desfavorables:

- Al iniciar el programa, e inicializar la Shield Ethernet (sin realizar aún conexión con el puente), la función **freeMemory()** devolvía un resultado de 7198 B, ejecutando todo esto en la función **setup()** propia de Arduino.
- Tras terminar la función **setup()**, y conectar con el puente Philips Hue, **freeMemory()** retornaba un valor de 6920 B.
- Finalmente, al concluir toda una iteración de la función **loop()** (la primera iteración), **freeMemory()** devolvía un valor de 2560 B, habiendo ahora realizado una petición al puente para recibir un JSON con información sobre todas las bombillas conectadas, y contando estas bombillas para obtener el número que había de ellas (ya que no hay ninguna forma propia en la API para realizar esta operación).

Justo después, al realizar la segunda iteración del **loop()**, la ejecución quedaba parada con la placa Arduino bloqueada, necesitando realizar un reset a esta para que volviese a iniciar su ejecución.

Tras dos meses de pruebas, corrección de errores y optimizaciones, y tras obtener estos resultados, esta solución fue descartada, debido a la gran falta de memoria de Arduino, que tenía que trabajar con grandes cadenas JSON de caracteres, dando paso a la solución final del Proyecto.

9.3.2. Raspberry Pi

Raspberry Pi es un computador embebido de bajo coste, el cual opera un sistema operativo GNU/Linux. En concreto, para este Proyecto se ha estudiado la posibilidad de utilizar una Raspberry Pi modelo B, al tener un rendimiento más que suficiente para la tarea que realizaría.

9.3.2.1. Características técnicas

La Raspberry Pi B, la versión más básica a la venta de Raspberry Pi con conexión Ethernet, cuenta con un procesador Broadcom BCM2835, un lector de tarjetas microSD en la cual se carga el sistema operativo y cuenta con las características mostradas en la tabla ?? [?].

Pines I/O digitales	26 pines GPIO.
Pines de entrada analógica	No.
Memoria	512 MB de memoria RAM, sin memoria flash. Los programas se ejecutan bajo el sistema operativo GNU/Linux.
Alimentación	MicroUSB.
Consumo	700 mA, 3.5 W.

Cuadro 9.5: Características técnicas principales de interés de Raspberry Pi B.

Gracias a ejecutar un sistema operativo GNU/Linux completo, este ordenador embebido es bastante completo y potente, pudiéndose ejecutar casi cualquier tipo de código y programas en él. Los pines GPIO de Raspberry Pi operan a 3.3 V, y utilizar un mayor voltaje derivaría en la destrucción del bloque GPIO y el SoC.

9.4. Electrónica para la detección de alertas

Para la detección de la señal enviada por el telefonillo, se han tenido en cuenta varias alternativas que surgen a raíz de la elección del tipo de telefonillo.

- La primera de las alternativas es utilizar un relé de 12 Vac con salida hasta 5 V, teniendo la salida conectada directamente al nodo central, y detectando las llamadas al telefonillo.
- La segunda alternativa propuesta es utilizar un convertor de 12 Vac a 5 V, realizando un sistema basado en un regulador de tensión o utilizando un convertor ya implementado que se encuentre en el mercado.

10. Descripción de la Solución Propuesta

10.1. Diseño de arquitectura de la solución

En la figura ?? se observa el Diagrama de bloques del sistema. El timbre conecta con el nodo central a través de un sistema de interconexión, el cual está compuesto por una placa PCB que monta un relé y conexionado. El telefonillo conecta de la misma manera con el nodo central, a través de otro sistema de interconexión. A su vez, el nodo central está conectado a un router, el cual conecta con el smartphone y con el puente de las bombillas utilizadas.

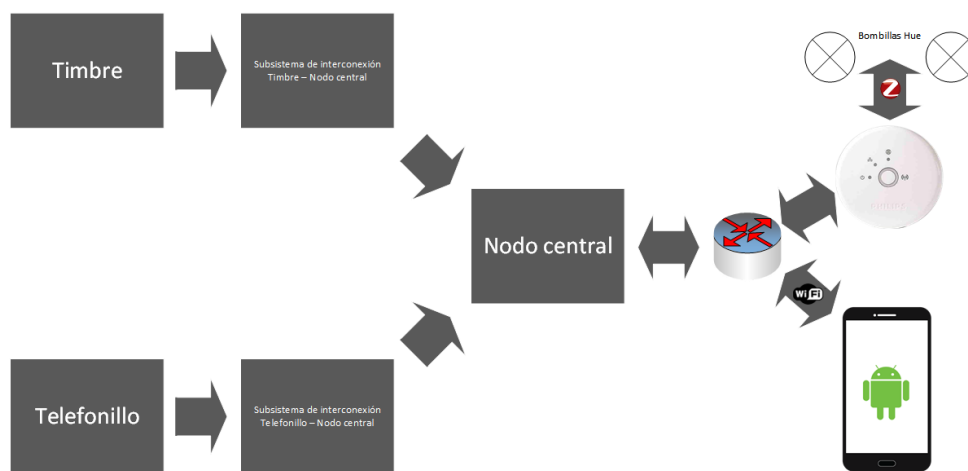


Figura 10.1: Diagrama de bloques del sistema.

Complementando este diseño, en el Anexo “Análisis y Diseño del Sistema”, apartado ??, se puede observar el diseño final del sistema de forma detallada, incluyendo todos los elementos que lo conforman.

10.2. Elección de soluciones

Una vez vistas todas las alternativas para cada caso en el capítulo anterior, es necesario elegir las alternativas que se adapten más a la solución pretendida para este Proyecto.

10.2.1. Elección de los nodos informativos visuales

Teniendo en cuenta la tabla ??, a raíz del estudio de Requisitos dentro de las Especificaciones del Sistema [??], y sobre todo, de acuerdo con el requisito no funcional NFR-01, se busca utilizar bombillas capaces de generar luces de colores (RGB), la segunda alternativa mostrada en la tabla queda descartada, pues tan solo proporciona luz en distintos tonos de blanco (ya sea luz más cálida o más fría, y variaciones de luminosidad).

Además, la tercera opción expuesta podría resultar a primera vista la solución más económica, pues permite la compra por separado del puente que comunica con las bombillas, pero debido a ser compra en el extranjero, con importantes gastos de transporte, y la imposibilidad de facturar estos gastos a través de la universidad, se descarta también esta opción.

Queda claro por tanto que la opción elegida es la primera, utilizar las bombillas Hue de Philips. Estas bombillas utilizan un puente ZigBee-Ethernet, al que se conecta mediante cable Ethernet y se utiliza una interfaz propia para alterar el estado de las bombillas. Estas se controlan mediante métodos HTTP a través de una interfaz RESTful, los cuales deben recibir y generar cadenas en formato JSON, por lo que es necesario poder manipular estas cadenas de alguna manera.

10.2.2. Elección de conectividad nodo central-smartphone

Se ha visto la necesidad de conectar un smartphone Android al sistema (según el requisito no funcional NFR-02), y la solución más acertada para la conexión de este dispositivo parece Wi-Fi, ya que como se ha visto, tiene mayor alcance que Bluetooth, pudiendo funcionar por tanto mucho mejor en el entorno del interior de una vivienda (teniendo en cuenta las paredes que se encuentran dentro de una vivienda).

Con Bluetooth, se tiene un alcance de 10 metros, 100 en condiciones óptimas (sin obstáculos) y mediante el uso de repetidores. Con Wi-Fi, el alcance es mucho mayor, llegando a 70-100 metros en interior, y hasta 250 metros en condiciones óptimas (en exteriores). Siendo estas dos las únicas opciones de conexión posibles para un smartphone Android, queda descartado el uso de Bluetooth, eligiendo así utilizar Wi-Fi como medio de comunicación.

Es necesario por tanto el uso de un router Wi-Fi al que el smartphone Android pueda conectar, para que se ubique en la misma red que el puente Hue de Philips. Por tanto, ambos dispositivos se interconectarán a través del router.

Por otra parte, el nodo central deberá poder conectar con esta red, y enviar los datos a ella. Se utilizará el acceso al router Wi-Fi, por lo que se descarta también el uso de Bluetooth para ello. Tanto Arduino como Raspberry Pi, las dos opciones analizadas para utilizar como nodo central, pueden lograr conexión a la red mediante un latiguillo Ethernet como solución más simple, necesitando Arduino de una placa Ethernet para lograr la conexión.

10.2.3. Elección de electrónica para la detección de alertas

Teniendo en cuenta la facilidad que supone el uso de un relé y la sencillez de su montaje electrónico e inclusión en el sistema, se opta por esta opción. Será necesario por tanto un relé de 12 Vac para la detección de la señal de llamada del telefonillo.

Además, se utilizará también un relé de 230 Vac para la detección de llamada al timbre de la puerta de la vivienda.

10.3. Solución final

Tras descartar Arduino como placa programable que funcionase como nodo central, se decidió continuar con la otra alternativa estudiada: utilizar una Raspberry Pi B como nodo central.

En la Raspberry Pi B se cuenta ya con interfaz de red Ethernet, por lo que no es necesario utilizar ninguna placa añadida. Además esta placa también cuenta con pines de entrada/salida programables, lo cual es indispensable para el desarrollo del Proyecto.

10.3.1. Topología y diseño de red

Para la implementación del sistema completo, se utilizará una topología de red en estrella, siendo el nodo central la Raspberry Pi, y los nodos subyacentes el telefonillo y el timbre (de los que recibirá información el nodo central), el puente Philips Hue, y el smartphone Android. De estos dos últimos se recibirá información además de enviar alertas hacia ellos.

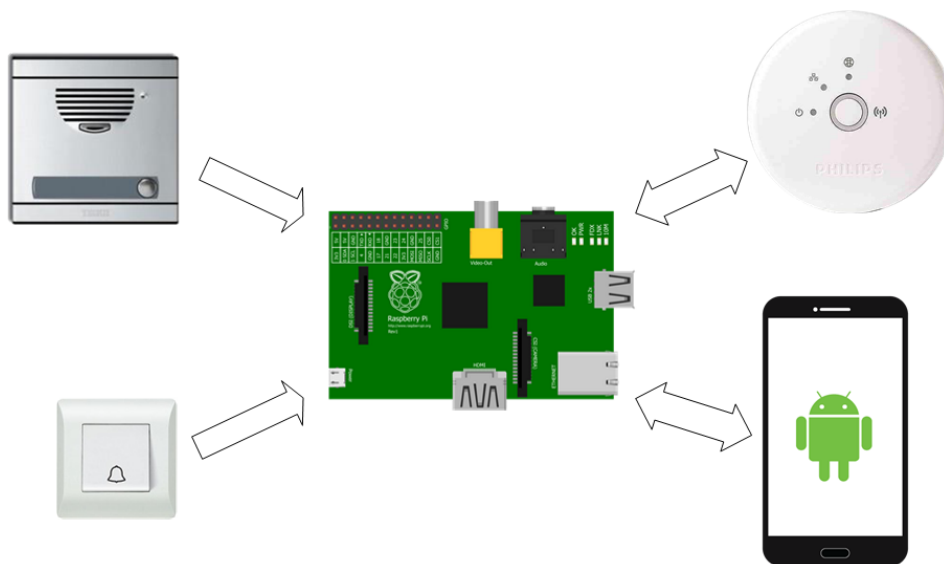


Figura 10.2: Diagrama de topología de red.

10.3.2. Implementación del nodo central

10.3.2.1. Sistema operativo

Existen varias alternativas, como son ArchLinux ARM, Raspbian o Minibian; pero se decide instalar como sistema operativo ArchLinux ARM, siendo más ligero que las variantes basadas en la distribución GNU/Linux Debian, al incluir un sistema más limpio de serie, sin entorno de escritorio ni servicios externos ya incluidos. Cualquiera de los 3 sistemas puede utilizarse para este Proyecto, siendo Raspbian el más pesado.

10.3.2.2. Desarrollo software del nodo central

Se utilizará Python como lenguaje de programación para los programas del sistema, lenguaje cada vez más utilizado en proyectos de este tipo en los que se necesita dotar a Raspberry Pi de funcionamiento junto a sensores y otros sistemas hardware. Además, Python incluye librerías que

permiten manejar cadenas JSON, establecer comunicaciones mediante peticiones HTTP, y más. Se utilizará la implementación 2.7 de Python, con soporte para todas las librerías necesarias en el Proyecto, como son las de manejo de direcciones URL.

Es necesario además instalar en el sistema operativo la librería RPi.GPIO, librería externa facilitada por Python para el manejo de los pines GPIO de Raspberry Pi, necesario para la lectura del estado de los relés empleados para la detección de llamadas al telefonillo o al timbre.

Es necesario, por tanto, desarrollar un conjunto de programas que sean capaces de comunicarse con el puente Philips Hue mediante métodos REST y cadenas JSON (utilizadas tanto para parámetros como para respuestas del puente), detectar las alertas de las llamadas entrantes tanto de telefonillo como de timbre a través de los relés empleados para ello, y comunicarse con un dispositivo Android para replicar las mismas alertas, a modo de texto.

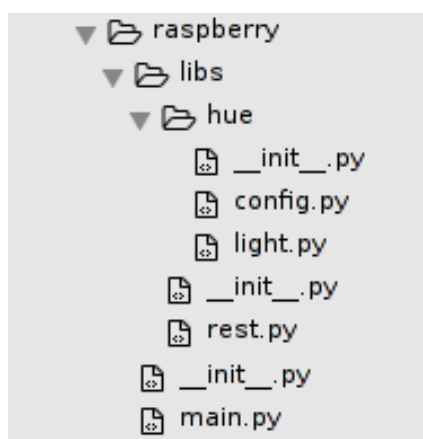


Figura 10.3: Estructura de directorios y ficheros del software del nodo central.

10.3.2.2.1. Librerías desarrolladas

Para ello, se ha desarrollado en primer lugar una librería REST en Python, que facilita la comunicación de Raspberry Pi con el puente Philips Hue. Esta librería, implementada en un único fichero, cuenta con 4 métodos principales, que serán los utilizados por el usuario: **get**, **post**, **put** y **delete**. Cada uno de ellos recibe la dirección del dispositivo al que realizar la petición, el dato a utilizar en caso de que sea necesario (POST y PUT), y el tipo de contenido del mensaje, que en este caso será siempre texto formateado en JSON.

```
1 import json
2 import urllib
3 import urllib2
4 class RestObject:
5
6     def get(self, url, data=None, content_type='application/json'):
7         return self._request(url, data, 'GET', content_type)
8
9     def post(self, url, data={}, content_type='application/json'):
```

MEMORIA

```

10     return self._request(url, data, 'POST', content_type)
11
12     def put(self, url, data={}, content_type='application/json'):
13         return self._request(url, data, 'PUT', content_type)
14
15     def delete(self, url, data=None, content_type='application/json'):
16         return self._request(url, data, 'DELETE', content_type)

```

Además, la misma clase de la librería cuenta con un método genérico privado el cual utilizan las 4 funciones anteriores. Este método es el verdadero encargado de establecer la conexión con el dispositivo deseado, y realizar la petición REST proporcionando además los datos que sean necesarios, y recogiendo la respuesta proporcionada por el dispositivo en formato JSON.

```

1  def _request(self, url, data, method, content_type):
2      if ((data or isinstance(data, dict)) and method not in ('GET', 'DELETE')):
3          # "data" is defined or is an instance of "dict" type.
4          data = json.dumps(data)
5          if (method == 'PUT'):
6              url_opener = urllib2.build_opener(urllib2.HTTPHandler)
7              request = urllib2.Request(url, data=data)
8              request.add_header('Content-Type', content_type)
9              request.get_method = lambda: 'PUT'
10             conn = url_opener.open(request)
11         else: # (method == 'POST')
12             headers = {'Content-Type': content_type}
13             try:
14                 request = urllib2.Request(url, data, headers)
15                 conn = urllib2.urlopen(request)
16             except TypeError:
17                 # POST without body.
18                 data = urllib.urlencode(data, 1)
19                 request = urllib2.Request(url, data, headers)
20                 conn = urllib2.urlopen(request)
21         elif (method == 'DELETE'):
22             url_opener = urllib2.build_opener(urllib2.HTTPHandler)
23             request = urllib2.Request(url)
24             request.get_method = lambda: 'DELETE'
25             conn = url_opener.open(request)
26         else: # (method == 'GET')
27             request = urllib2.Request(url)
28             conn = urllib2.urlopen(request)
29         response = conn.read()
30         conn.close()
31         try:
32             content = json.loads(response)
33         except:
34             content = response
35         return content

```

Lo primero que hace este método es comprobar si hay datos entrantes, si los hay, los formatea como JSON mediante la librería propia de Python, y luego comprueba el tipo de petición (PUT o POST, las dos peticiones que pueden recibir datos entrantes).

Para efectuar el PUT, establece la conexión utilizando la dirección recibida, añade los datos a

enviar, y la cabecera con el tipo de contenido. Finalmente realiza la petición. Por otro lado, para POST, se distinguen dos tipos: con y sin cuerpo del mensaje. Se realizan los mismos pasos, salvo que efectuando una petición POST.

En cambio, si no hay datos entrantes, la petición podrá ser GET o DELETE. Se realizan las mismas acciones que en el caso anterior, solo que sin incluir datos en la petición. Finalmente, se almacena la respuesta, se cierra la conexión y se devuelve la respuesta.

Por otra parte, se ha creado una librería Python para el control del puente Philips Hue, así como de las bombillas asociadas a él. Gracias a esta librería, se evita utilizar la interfaz web propia del puente, que únicamente debe utilizarse como herramienta para desarrolladores, en concreto para depuración y pruebas. Esta librería se divide en tres clases:

- Bridge: clase principal, de la que se debe crear un objeto para poder utilizar el sistema. Contendrá información como el usuario del puente Philips Hue a utilizar, o la dirección IP del mismo.
- Light: clase que implementa todas las funciones necesarias para el control de las bombillas, como obtener su estado o modificarlo.
- Config: clase que contiene funciones necesarias para la configuración del puente Philips Hue desde el mismo programa, sin necesidad de acceder a la interfaz web de depuración.

Estas clases están repartidas en distintos archivos, estando además la clase principal implementada en el fichero `__init__.py` del paquete de la librería, siguiendo las directrices de Python para la implementación de clases, métodos y otras funciones que se deseen ejecutar como inicialización de las librerías o paquetes creados. Esta clase principal, Bridge, cuenta únicamente con un constructor.

```
1 class Bridge():
2
3     def __init__(self, device, user):
4         from config import Config
5         from light import Light
6
7         self.config = Config(device, user)
8         self.light = Light(device, user)
9
10        # After create Light object, search for new lights.
11        self.light.findNewLights()
```

El constructor recibe la dirección IP del puente y el nombre del usuario del puente a utilizar, crea un objeto de cada una de las otras clases, Config y Light, utilizando estos parámetros; y busca en la misma inicialización bombillas que hayan sido recientemente instaladas en la vivienda, para asociarlas al puente Hue.

La clase Config únicamente contiene 3 métodos relacionados con la configuración y estado del puente. Esta clase además, utiliza la librería REST anteriormente expuesta, para la comunicación con el puente.

MEMORIA

```

1
2 from libs.rest import RestObject
3
4 class Config:
5
6     def __init__(self, bridge, user):
7         self.bridge = bridge
8         self.user = user
9
10    def isConnected(self):
11        rest_obj = RestObject()
12        path = 'api/{username}'.format(username=self.user['name'])
13        url = 'http://{bridge_ip}/{path}'.format(bridge_ip=self.bridge['ip'], path=path)
14        response = rest_obj.get(url)
15        response = dict(resource=response)
16        if 'lights' in response['resource']:
17            return True
18        elif 'error' in response['resource'][0]:
19            error = response['resource'][0]['error']
20            if error['type'] == 1:
21                return False

```

Obviando el constructor, el cual es trivial, el primero de los métodos es un observador que comprueba si se puede conectar al puente con el usuario dado, o por el contrario, es un usuario no autorizado a utilizar dicho dispositivo. Este método realiza una petición GET a la dirección “*http://IP_PUENTE/api/USUARIO*”, recibiendo como respuesta una cadena JSON con toda la configuración del puente si el usuario está autorizado, o un error si el usuario no está dentro de la lista de usuarios autorizados a realizar operaciones con el puente.

```

1    def createUser(self, user):
2        rest_obj = RestObject()
3        url = 'http://{bridge_ip}/api'.format(bridge_ip=self.bridge['ip'])
4        resource = {'devicetype': user, 'username': user}
5        response = rest_obj.post(url, resource)
6        return dict(resource=response)

```

El segundo método recibe una cadena con el nombre de un nuevo usuario, y crea un usuario en el puente con ese nombre. De nuevo, utilizando peticiones REST, en concreto la petición POST, se realiza la operación, resultando una cadena JSON que informará si la operación se ha realizado con éxito.

```

1    def deleteUser(self, user):
2        rest_obj = RestObject()
3        service = 'config/whitelist/{id}'.format(id=user)
4        path = 'api/{username}/{service}'.format(username=self.user['name'], service=
            service)
5        url = 'http://{bridge_ip}/{path}'.format(bridge_ip=self.bridge['ip'], path=path)
6        response = rest_obj.delete(url)
7        return dict(resource=response)

```

Por último, el tercer método realiza justo lo contrario que el anterior: elimina un usuario de la lista de usuarios autorizados a utilizar el puente, recibiendo como parámetro el nombre del usua-

rio a eliminar. Mediante una petición DELETE a la dirección necesaria, se elimina ese usuario, y se devuelve información relativa al resultado de la operación.

La última clase de la librería Python para Hue es Light. Esta clase implementa las funciones necesarias para controlar las bombillas Hue, como son obtener el estado de las bombillas, el número de bombillas conectadas, el estado de una bombilla individual, encontrar nuevas bombillas que conectar al puente Hue, y diversos modificadores de su estado.

```
1 from libs.rest import RestObject
2
3 class Light:
4
5     def __init__(self, bridge, user):
6         self.bridge = bridge
7         self.user = user
8
9     def get(self, resource={'which': 'all'}):
10         rest_obj = RestObject()
11         services = {
12             'all': {'service': 'lights'},
13             'new': {'service': 'lights/new'}
14         }
15         if isinstance(resource['which'], int):
16             resource['id'] = resource['which']
17             resource['which'] = 'one'
18         if (resource['which'] == 'one'):
19             services['one'] = {'service': 'lights/{id}'.format(id=resource['id'])}
20         service = services[resource['which']]['service']
21         path = 'api/{username}/{service}'.format(username=self.user['name'], service=
            service)
22         url = 'http://{bridge_ip}/{path}'.format(bridge_ip=self.bridge['ip'], path=path)
23         response = rest_obj.get(url)
24         if service == 'lights':
25             lights = []
26             for (k, v) in response.items():
27                 v['id'] = int(k)
28                 lights.append(v)
29             response = lights
30         return dict(resource=response)
```

Al igual que en la clase Config, el constructor es bastante trivial. El primer método de interés que se encuentra es **get**, el cual se encarga de obtener del puente Philips Hue la información de todas las bombillas conectadas, las nuevas, o una única bombilla determinada, dependiendo de la información que reciba como parámetro. Una vez controlado de qué bombilla o bombillas debe obtener la información, realiza una petición GET, y devuelve el resultado de esta operación, exceptuando si debe obtener información de todas las bombillas, caso en el que reagrupa la información de las bombillas en un vector de bombillas (vector de información de cada una de las bombillas).

MEMORIA

```

1  def getNumLights(self):
2      rest_obj = RestObject()
3      path = 'api/{username}/lights'.format(username=self.user['name'])
4      url = 'http://{bridge_ip}/{path}'.format(bridge_ip=self.bridge['ip'], path=path)
5      response = rest_obj.get(url)
6      lights = []
7      for (k, v) in response.items():
8          v['id'] = int(k)
9          lights.append(v)
10     return len(lights)

```

El método **getNumLights** se basa en el método anterior, y a diferencia de este no recibe parámetros. Este método realiza directamente una petición GET para obtener el estado de todas las bombillas conectadas al puente, para después transformar la respuesta en un vector de bombillas, y finalmente devolver la longitud de ese vector (o lo que es lo mismo, el número de bombillas).

```

1  def getLightState(self, light):
2      return self.get({'which':light})['resource']['state']
3
4  def isPhisicallyOn(self, light):
5      return self.get({'which':light})['resource']['state']['reachable']

```

Los siguientes métodos, **getLightState** y **isPhisicallyOn** reciben como parámetro el número identificador de una bombilla, y utilizan internamente el método **get** expuesto anteriormente. El primero, filtra la salida de **get**, devolviendo únicamente el campo **state** del JSON generado. El segundo, hace lo mismo y además se queda solo con el parámetro **reachable** de **state**, el cual determinar si una bombilla está encendida físicamente o no (y no a través del puente Hue).

```

1  def findNewLights(self):
2      rest_obj = RestObject()
3      path = 'api/{username}/lights'.format(username=self.user['name'])
4      url = 'http://{bridge_ip}/{path}'.format(bridge_ip=self.bridge['ip'], path=path)
5      response = rest_obj.post(url)
6      return dict(resource=response)

```

La función **findNewLights** realiza una operación parecida a **get**, cambiando que ejecuta una petición POST en lugar de GET. El resultado de realizar POST a la misma dirección mediante la cual se obtiene información de todas las bombillas con **get**, es buscar nuevas bombillas pendientes de conectar al puente, y asociarlas automáticamente a este.

```
1 def update(self, resource):
2     rest_obj = RestObject()
3     if (resource['data'].has_key('attr')):
4         service = 'lights/{id}'.format(id=resource['which'])
5         data = resource['data']['attr']
6     elif (resource['data'].has_key('state')):
7         service = 'lights/{id}/state'.format(id=resource['which'])
8         data = resource['data']['state']
9     else:
10        raise Exception('Unknown data type.')
11    path = 'api/{username}/{service}'.format(username=self.user['name'], service=
12        service)
13    url = 'http://{bridge_ip}/{path}'.format(bridge_ip=self.bridge['ip'], path=path)
14    response = rest_obj.put(url, data)
15    return dict(resource=response)
```

Utilizado por los modificadores de estado como método base, el método **update** es un método genérico para modificar el estado de una bombilla conectada al puente Hue. Este método recibe un dato, y dependiendo de si ese dato es interno al parámetro **attr** o **state** de la bombilla, realiza la petición POST a distintas URL, junto al dato que se ha proporcionado. El resultado de este método es la modificación del estado de la bombilla, además de devolver información sobre la operación realizada.

```
1 def setLightColor(self, light, bri, hue, sat):
2     resource = {
3         'which':light,
4         'data':{'state':{'bri':bri,'hue':hue,'sat':sat}}
5     }
6     return self.update(resource)
7
8 def setLightState(self, light, on, bri, hue, sat):
9     resource = {
10        'which':light,
11        'data':{'state':{'on':on,'bri':bri,'hue':hue,'sat':sat}}
12    }
13    return self.update(resource)
14
15 def setLightOn(self, light):
16     resource = {
17         'which':light,
18         'data':{'state':{'on':True}}
19     }
20     return self.update(resource)
21
22 def setLightOff(self, light):
23     resource = {
24         'which':light,
25         'data':{'state':{'on':False}}
26     }
27     return self.update(resource)
```

Los últimos métodos de la clase, los modificadores de estado, son bastante simples, al utilizar internamente el método **update** genérico anterior. **setLightColor** recibe el identificador de una bombilla y los parámetros relacionados con el color, y realiza un **update** para modificar

MEMORIA

estos parámetros. **setLightState** realiza lo mismo y además, modifica también si la bombilla quedará encendida o apagada, recibiendo este atributo de estado como parámetro. Por otra parte, **setLightOn** y su contrapartida, **setLightOff** sirven para alterar el estado de encendido de la bombilla: encenderla y apagarla, respectivamente.

10.3.2.2.2. Programa principal

El programa principal del nodo central utiliza las librerías anteriores para conectar con el puente Philips y controlar las bombillas. Este programa además utiliza también varias librerías de Python como son **socket**, **sys**, **threading**, y la librería externa **RPi.GPIO**, entre otras.

```

1 #####
2 # Configurable zone:
3 USERNAME = 'ucahueuser'
4 DING=17
5 INTERCOM=18
6 LANG='es'
7 # End of configurable zone.
8 #####
9
10 # GPIO setup:
11 GPIO.setmode(GPIO.BCM)
12 GPIO.setwarnings(False)
13 GPIO.setup(DING,GPIO.IN)
14 GPIO.setup(INTERCOM,GPIO.IN)
15
16 # Used colors:
17 RED=0
18 BLUE=46920
19
20 # Global vars without initial value:
21 data=""
22 bridge=None
23 sock=None
24
25 # Command to discover Philips Hue bridge's IP.
26 cmd = 'ip route show | grep "src" | egrep -o "([0-9]{1,3}\.
      {3}[0-9]{1,3}/[0-9]{1,2}" | xargs nmap -sn | grep Philips-hue | egrep -o
      "([0-9]{1,3}\.){3}[0-9]{1,3}"'
27
28 # Messages vars:
29 MSG_LINK = ""
30 MSG_LINK_ES ="Presiona el botón del puente Hue."
31 MSG_LINK_EN ="Press the button on the Hue bridge."
32 MSG_DING = ""
33 MSG_DING_ES = "Alguien llama al timbre."
34 MSG_DING_EN = "Somebody rings the bell."
35 MSG_INTERCOM = ""
36 MSG_INTERCOM_ES = "Alguien llama al telefonillo."
37 MSG_INTERCOM_EN = "Somebody calls the intercom."

```

En la primera parte del programa, se definen varias variables utilizadas a lo largo de este. Las cuatro primeras variables pueden modificarse, definiéndose en ellas el nombre de usuario a utilizar para conectar al puente, el idioma de los mensajes enviados al smartphone (español o inglés),

y el número de los pines GPIO a los que se conectarán las señales del telefonillo y el timbre.

El resto de variables definidas son variables de cadenas de caracteres en los distintos idiomas, el identificador Hue de los colores a utilizar para mostrar las alertas mediante las bombillas, la configuración de los pines GPIO, y el comando a utilizar para descubrir la dirección IP asociada al puente Hue de manera automática. Este último utiliza comandos de consola del sistema para obtener la dirección IP de la red a la que se está conectado mediante **ip route show** y varios **grep** (uno de ellos utilizando expresiones regulares para determinar el formato de la dirección IP), proporcionar ese comando a la herramienta de escaneo de redes **nmap**, la cual proporcionará la dirección IP de los dispositivos conectados a la red, filtrando por último el dispositivo de interés (el puente Philips Hue) para obtener únicamente su dirección IP.

```
1 def linkUserConfig():
2     global data
3     created = False
4     print MSG_LINK
5     data = MSG_LINK
6     while not created:
7         response = bridge.config.createUser(USERNAME) ['resource']
8         if 'error' in response[0]:
9             if response[0]['error']['type'] != 101:
10                print 'Unhandled error creating configuration on the Hue'
11                sys.exit(response)
12            else:
13                created = True
```

Entre los métodos de este programa se encuentra **linkUserConfig**, el cual se encarga de asociar el usuario configurado al puente Philips Hue si no estuviese ya creado y admitido dentro de la lista de usuarios con permisos para controlar el puente, mediante la función **createUser** de la clase Config.

```
1 def lightAlert(alert_color):
2     numlights = bridge.light.getNumLights()
3     oldstate = dict()
4     for i in range(1,numlights+1):
5         oldstate[i] = bridge.light.getLightState(i)
6     for i in range(1,4):
7         for j in range(1,numlights+1):
8             if bridge.light.isPhisicallyOn(j):
9                 bridge.light.setLightState(j, True, 254, alert_color, 254)
10            time.sleep(1)
11            for j in range(1,numlights+1):
12                if bridge.light.isPhisicallyOn(j):
13                    bridge.light.setLightColor(j, oldstate[j]['bri'], oldstate[j]['hue'], oldstate[j]['sat'])
14            time.sleep(0.3)
15            for j in range(1,numlights+1):
16                if bridge.light.isPhisicallyOn(j) and not oldstate[j]['on']:
17                    bridge.light.setLightOff(j)
18            time.sleep(1)
```

El método para mostrar las alertas a modo de parpadeo de luces de colores en las bombillas es

MEMORIA

lightAlert, el cual recibe el identificador de color de la alerta, y realiza una serie de operaciones para almacenar el estado previo de todas las bombillas conectadas al puente (mediante el método **getLightState** de la clase **Light**), cambia su estado a uno de encendido y con el color suministrado mediante parámetro, y vuelve a colocar las bombillas en su estado anterior (teniendo en cuenta si alguna de ellas estaba anteriormente apagada). Esta operación se realiza varias veces para dar un efecto de parpadeo a la alerta.

```

1 def sendAlertToAndroid():
2     while True:
3         global data
4         req, client_address = sock.recvfrom(1024) # get the request, 1kB max
5         # Look in the first line of the request for a valid command
6         # The command should be 'http://server/getAlert'
7         match = re.match('GET /getAlert', req)
8         if match:
9             if data != "":
10                sock.sendto(data, (client_address[0], 8081))
11                data = ""
12            else:
13                sock.sendto("", (client_address[0], 8081))
14        else:
15            # If there was no recognised command then return a 404 (page not found)
16            print "Returning 404"
17            sock.sendto("HTTP/1.1 404 Not Found\r\n", (client_address, 8081))

```

El último de los métodos secundarios del programa principal, **sendAlertToAndroid** es el encargado de quedar a la espera de conexiones entrantes mediante el socket establecido en el método principal del programa. Si se recibe una petición, se controla que es una petición válida (deberá contener el texto “/getAlert”), y se envían los datos definidos en el programa hacia el smartphone, utilizando la dirección IP que realizaba la petición, y el puerto 8081. Si la petición no es válida, se generará un mensaje de error y se enviará de la misma manera.

Es ya en la función **main()** donde se realizan los últimos ajustes e inicializaciones para que el sistema se ponga en marcha, y donde se desarrolla la ejecución principal del programa.

```

1 def main():
2     global data
3     global bridge
4     global sock
5
6     # UDP socket configuration:
7     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
8     sock.bind(('', 8080))
9
10    # Gets Hue bridge's IP and create an Bridge object.
11    HUE_IP = os.popen(cmd).read().strip()
12    bridge = Bridge(device={'ip':HUE_IP}, user={'name':USERNAME})
13
14    # Sets message vars.
15    if LANG == 'es':
16        locale.setlocale(locale.LC_TIME, "es_ES.utf8")
17        MSG_LINK = MSG_LINK_ES
18        MSG_DING = MSG_DING_ES

```

```
19     MSG_INTERCOM = MSG_INTERCOM_ES
20 else:
21     locale.setlocale(locale.LC_TIME, "")
22     MSG_LINK = MSG_LINK_EN
23     MSG_DING = MSG_DING_EN
24     MSG_INTERCOM = MSG_INTERCOM_EN
25
26 t = threading.Thread(target=sendAlertToAndroid)
27 t.start()
28 while True:
29     data = ""
30     if not bridge.config.isConnected():
31         print 'Unauthorized user'
32         linkUserConfig()
33
34     if not GPIO.input(INTERCOM):
35         cad = MSG_INTERCOM
36         data = time.strftime("%d/%b %H:%M:%S")+" - "+cad
37         lightAlert(RED)
38
39     if not GPIO.input(DING):
40         cad = MSG_DING
41         data = time.strftime("%d/%b %H:%M:%S")+" - "+cad
42         lightAlert(BLUE)
43
44     bridge.light.findNewLights()
```

En este método se crea un socket UDP en el puerto 8080, el cual será el encargado de recibir peticiones de nuevas alertas desde el smartphone, y enviar estas alertas hacia el dispositivo.

A continuación, se obtiene la dirección IP del puente Hue con la secuencia de comandos anteriormente descrita, y se crea un objeto de la clase Bridge (el cual creará a su vez una instancia de las clases Config y Light), para poder comunicar este programa con el puente. Tras ajustar las variables de mensajes al idioma seleccionado, se inicia un hilo que tendrá como objetivo el método **sendAlertToAndroid()**.

Una vez que está listo todo lo anterior, empieza la iteración sin pausa de la parte principal de este programa, donde se comprueba si el usuario tiene permisos de conexión al puente mediante el método **isConnected** de la clase Config. Si no los tiene, se utilizará el método **linkUserConfig()** de este mismo programa.

Tras esta comprobación, la cual se realiza en cada ejecución por si el usuario a utilizar fuese eliminado de la lista de usuarios permitidos, se comprueban los pines GPIO asociados al timbre y al telefonillo, con el objetivo de detectar alertas generadas por estos. De realizar una lectura positiva de alerta, se ejecuta la función **lightAlert** con el color asociado al tipo de alerta, estableciendo como parámetro la fecha y hora de la alerta, junto con una cadena de texto a mostrar en el smartphone.

Finalmente, al igual que se realiza al principio de la ejecución del programa al crear el objeto Bridge, se comprueba si se pueden asociar nuevas bombillas al puente Hue, para tener control en todo momento sobre todas las bombillas al alcance del puente.

10.3.3. Aplicación para Android

10.3.3.1. Diseño de la aplicación

El diseño de la aplicación sigue las normas de diseño de Google, utilizando una interfaz basada en MaterialDesign. Tras realizar un diseño de prototipo, el cual se puede observar en el apartado ??, previo al uso del IDE Android Studio, se decide basar el estilo general de la aplicación en el predeterminado generado por Android Studio al utilizar MaterialDesign como base. En cada pantalla de la aplicación se encuentra la barra superior de la aplicación, ActionBar, en la que se encuentra el acceso al menú de configuración o el botón de vuelta a la pantalla anterior, según la vista en la que se encuentre el usuario. En la figura ?? se puede observar la pantalla principal de la aplicación.

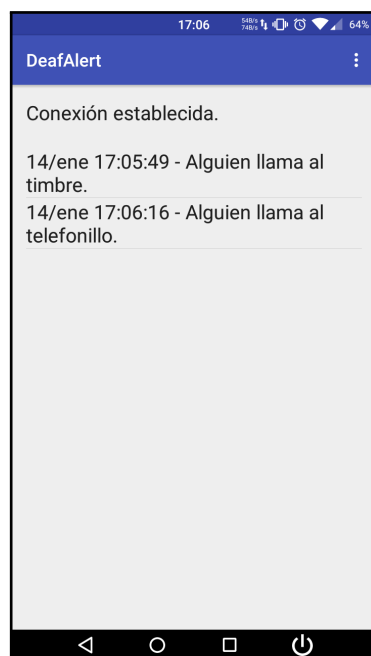


Figura 10.4: Pantalla principal de la aplicación para Android.

En esta pantalla principal se encuentra la lista de alertas recibidas desde el servidor de alertas (la Raspberry Pi), además de un texto que indica el estado de conexión o de alertas entrantes. Tal como se muestra en la figura ??, estas alertas pueden ser eliminadas a mano desde la propia aplicación, de tal manera que cuando el usuario pulsa en una de ellas, la aplicación mostrará un mensaje preguntando si se desea eliminar la alerta.

Además de esta pantalla principal, la aplicación cuenta con un menú de configuración, fácilmente accesible mediante el menú desplegable superior, donde se puede configurar la dirección IP del servidor de alertas, tal y como se observa en la figura ??.

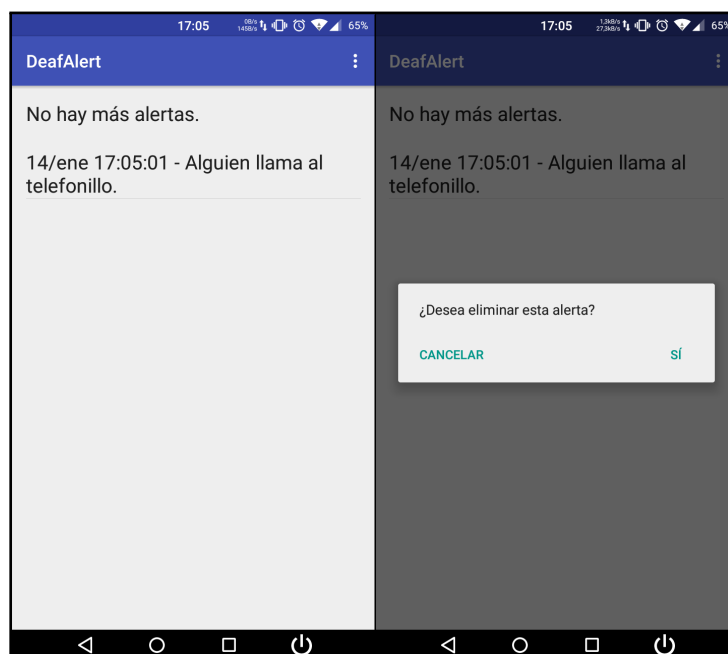


Figura 10.5: Borrado de alertas en aplicación para Android.

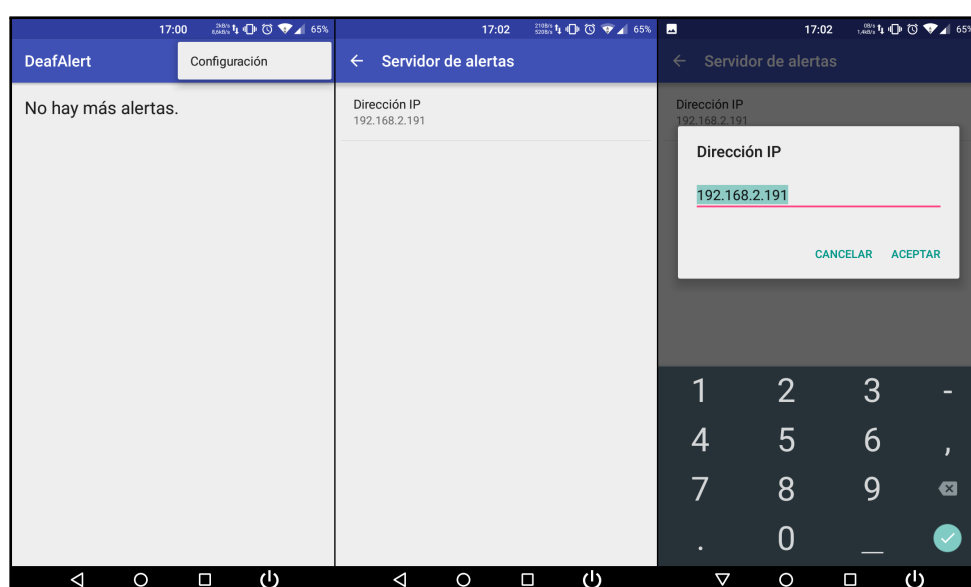


Figura 10.6: Menú de configuración de aplicación para Android.

10.3.3.2. Estructura principal

La aplicación Android, desarrollada con el IDE Android Studio, se basa en tres clases java principales, divididas en archivos del mismo nombre:

- **MainActivity**: Clase principal de la aplicación, encargada de construir la vista principal de la aplicación Android, y dar funcionalidad a la misma. Es además la encargada de comunicar con el nodo central, tratar las alertas y mostrarlas en la pantalla principal.
- **AlertListAdapter**: Clase utilizada por MainActivity, para controlar la lista de alertas entrantes e ir refrescando la vista principal de la aplicación con esas alertas.
- **SettingsActivity**: Clase generada por el propio IDE, la cual implementa el menú de configuración de la aplicación, modificada para dotar a la aplicación de un menú de configuración que se adapte a las necesidades.

Además de estas clases java, la aplicación cuenta con otra clase java más generada por Android Studio, la cual no se analizará, pues es código generado automáticamente.

10.3.3.2.1. Actividad principal

En la actividad principal (MainActivity) es donde se desarrolla la funcionalidad principal de la aplicación, y es la encargada de generar la vista principal de la misma (la vista a la que se accede una vez se abre la aplicación).

```

1 public class MainActivity extends AppCompatActivity {
2
3     private InetAddress raspberryAddr;
4     private ListView alertList;
5     private ArrayList<String> alerts;
6     private AlertListAdapter alertAdapter;
7     private TextView textRaspberry;
8     private int NotifCount;
9     private boolean appIsInForegroundMode;
10
11
12     @Override
13     protected void onCreate(Bundle savedInstanceState) {
14         super.onCreate(savedInstanceState);
15         setContentView(R.layout.activity_main);
16         Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
17         setSupportActionBar(toolbar);
18         textRaspberry = (TextView) findViewById(R.id.textRaspberry);
19         alertList = (ListView) findViewById(R.id.alertList);
20         alerts = new ArrayList<>();
21         alertAdapter = new AlertListAdapter(MainActivity.this, alerts);
22         alertList.setAdapter(alertAdapter);
23         NotifCount = 0;
24         appIsInForegroundMode = false;
25         Timer timer = new Timer();
26         timer.scheduleAtFixedRate(new TimerTask() {
27             @Override
28             public void run() {
29                 printRaspberryData();
30             }

```

```

31         }, 0, 1500);
32         StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().
            permitAll().build();
33         StrictMode.setThreadPolicy(policy);
34     }

```

Esta clase cuenta con varios atributos necesarios para almacenar la dirección IP del servidor de alertas, referencias a partes de la vista principal, una lista de cadenas que constituyen las alertas, un adaptador para estas alertas que será el encargado de refrescar las vistas, y otros atributos utilizados para control de notificaciones.

La función principal de la clase y por tanto de la actividad es el método **onCreate**, el cual se encarga de dotar de funcionalidad a la aplicación cuando la vista principal es cargada. Además de inicializar algunos de los atributos anteriormente citados con valores predeterminados establecidos en la aplicación, se establece un objeto **AlertListAdapter** asociado a la actividad y a la lista de alertas. Una vez que esto está hecho, se crea un objeto de la clase **Timer**, el cual se encarga de ejecutar un método determinado cada cierto periodo de tiempo. Este método es **printRaspberryData**, el cual se examinará más adelante.

Con todos los atributos inicializados y el **Timer** puesto en marcha, se establecen las políticas de la aplicación y se pone esta en funcionamiento.

```

1     private void printRaspberryData() {this.runOnUiThread(printAlerts);}
2
3     private Runnable printAlerts = new Runnable() {
4         @Override
5         public void run() {
6             ConnectivityManager connManager = (ConnectivityManager) getSystemService(
7                 Context.CONNECTIVITY_SERVICE);
8             NetworkInfo wifi = connManager.getNetworkInfo(ConnectivityManager.TYPE_WIFI
9             );
10            boolean stop = false;
11            if (wifi != null && wifi.isConnected()) {
12                SharedPreferences sharedPref = PreferenceManager.
13                    getDefaultSharedPreferences(MainActivity.this);
14                String raspberry = sharedPref.getString("ip_address", getText(R.string.
15                    pref_default_ip_address).toString());
16
17                DatagramSocket socket;
18                try {
19                    raspberryAddr = InetAddress.getByName(raspberry);
20                    socket = new DatagramSocket(8081);
21                    socket.connect(raspberryAddr, 8080);
22                    if(socket.isConnected()){
23                        byte[] reqBuff = "GET /getAlert".getBytes();
24                        DatagramPacket reqPacket = new DatagramPacket(reqBuff, reqBuff.
25                            length, raspberryAddr, 8080);
26                        try {
27                            socket.send(reqPacket);
28                        } catch (IOException e) {
29                            e.printStackTrace();
30                            stop = true;
31                            textRaspberry.setText(getText(R.string.
32                                mainactivity_send_packet_error));
33                        }
34                    }
35                }
36            }
37        }
38    }

```

```

27         }
28         if(!stop) {
29             boolean stop2 = false;
30             byte[] recvBuff = new byte[1024];
31             DatagramPacket recvPacket = new DatagramPacket(recvBuff,
32                 recvBuff.length);
33             try {
34                 socket.receive(recvPacket);
35             } catch (IOException e) {
36                 textRaspberry.setText(getText(R.string.
37                     mainactivity_no_data_error));
38                 e.printStackTrace();
39                 stop2 = true;
40             }
41             if (!stop2) {
42                 String cad = new String(recvPacket.getData(), 0, recvPacket
43                     .getLength());
44                 if (!cad.equals("")) {
45                     textRaspberry.setText(getText(R.string.
46                         mainactivity_conn_ok));
47                     alertAdapter.setData(cad);
48                     if (!appIsInForegroundMode) notifyAlertToNotification(cad
49                         );
50                 }else textRaspberry.setText(getText(R.string.
51                     mainactivity_no_data_error));
52             }
53             socket.close();
54             } //stop
55         }else{
56             textRaspberry.setText(getText(R.string.mainactivity_conn_error));
57             socket.close();
58         }
59         socket.close();
60     } catch (Exception e) {
61         textRaspberry.setText(getText(R.string.mainactivity_server_conn_error
62             ));
63         e.printStackTrace();
64     }
65 } else {
66     textRaspberry.setText(getText(R.string.mainactivity_wifi_error));
67 }
68 }
69 };

```

El método **printRaspberryData** tiene un funcionamiento bastante sencillo, el cual se limita a establecer un objeto de tipo **Runnable** para el hilo principal de la interfaz de la aplicación.

Es en este objeto **Runnable** donde se desarrolla el funcionamiento principal de la aplicación. En él se establece el comportamiento del método **run()**, método asociado a cada ejecución del hilo. Lo primero que realiza este método es comprobar la conexión WiFi del dispositivo. Si este no está habilitado, muestra un mensaje de error y sale del método, esperando a la próxima ejecución de este método a raíz del **Timer** establecido en **onCreate** para comprobar de nuevo si la conexión WiFi ha sido habilitada.

Una vez el dispositivo cuente con conexión WiFi, se obtienen las preferencias almacenadas en el contenedor `SharedPreferences` de la aplicación. De estas preferencias se obtiene la dirección IP del servidor de alertas del que recibir las mismas.

A continuación, se crea un objeto **DatagramSocket**, un socket de conexión UDP, el cual se configura para conectar a la dirección del servidor de alertas (la Raspberry Pi) a través del puerto 8080. Además se establece el puerto de escucha de datos entrantes en el puerto 8081. Si la conexión se lleva a cabo correctamente, se realiza una petición con el dato asociado “GET /getAlert”, incluyendo este mensaje en un objeto **DatagramPacket**, paquete UDP que será enviado a través del socket.

Si el envío surte efecto, se crea un nuevo objeto `DatagramPacket`, pero esta vez no será enviado, sino que servirá para almacenar los datos entrantes. A continuación, el hilo queda a la espera de alguna conexión entrante a través del socket. Una vez hay datos entrantes, el paquete se almacena en el objeto creado anteriormente, se extrae el dato (la alerta), y se añade a la lista de alertas, mostrándola por pantalla a través del objeto **AlertListAdapter**. Además, se muestra un mensaje de conexión satisfactoria en la aplicación, y si la aplicación estuviera en segundo plano o el dispositivo se encontrase en modo reposo (bloqueado), se envía una notificación al sistema con la alerta entrante. El comportamiento del objeto `AlertListAdapter`, así como lo que ocurre con la lista de alertas, se tratan en el apartado ??.

```

1      @Override
2      protected void onResume() {
3          super.onResume();
4          appIsInForegroundMode = true;
5          alertList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
6              @Override
7              public void onItemClick(AdapterView<?> parent, View view, final int
8                  position, long id) {
9                  AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this
10                      );
11                  builder.setMessage(getText(R.string.question))
12                      .setCancelable(true)
13                      .setNeutralButton(getText(R.string.cancel), new DialogInterface.
14                          OnClickListener() {
15                              public void onClick(DialogInterface dialog, int id) {
16                                  dialog.cancel();
17                              }
18                          })
19                      .setPositiveButton(getText(R.string.okey), new DialogInterface.
20                          OnClickListener() {
21                              public void onClick(DialogInterface dialog, int id) {
22                                  alertAdapter.remove(position);
23                              }
24                          });
25                  AlertDialog alert = builder.create();
26                  alert.show();
27              }
28          });
29          registerContextMenu(alertList);
30      }

```

Además, en el método **onResume** se establece el comportamiento de la app cuando se pulse en alguna de las alertas de la lista, utilizando para ello el método **OnItemClickListener** de la clase **AdapterView**, el cual crea una instancia de un objeto de escucha de clicks en items o elementos de la lista. Redefiniendo el método **onItemClick** se establece este comportamiento. En primer lugar, se crea un objeto **AlerDialog.Builder**, el cual se encargará de generar un diálogo emergente de alerta. A continuación se establecen el mensaje del diálogo (mediante el método **setMessage**), la funcionalidad al pulsar el botón “Cancelar”, el cual simplemente saldrá del diálogo de alerta, y la funcionalidad del botón “Sí”, el cual eliminará la alerta de la lista. Finalmente, se genera el objeto de alerta y se muestra la alerta mediante el método **show**.

```

1      @Override
2      public boolean onCreateOptionsMenu(Menu menu) {
3          // Inflate the menu; this adds items to the action bar if it is present.
4          getMenuInflater().inflate(R.menu.menu_main, menu);
5          return true;
6      }
7
8      @Override
9      public boolean onOptionsItemSelected(MenuItem item) {
10         switch (item.getItemId()) {
11             case R.id.settings_menu:
12                 Intent intent = new Intent(this, SettingsActivity.class);
13                 this.startActivity(intent);
14                 break;
15             default:
16                 return super.onOptionsItemSelected(item);
17         }
18         return true;
19     }

```

Por otro lado, mediante el método **onCreateOptionsMenu** se establece el menú de la aplicación, generando un botón para acceder al menú mediante el método **getMenuInflater**.

Además, se dota de funcionalidad al menú con el método **onOptionsItemSelected**, donde se asocia cada elemento del menú con una actividad a iniciar (una nueva vista de la aplicación). Esto se realiza mediante el uso de objetos de la clase **Intent**, asociando en este caso una instancia de esta clase a la clase **SettingsActivity**, clase que implementa la actividad del menú, que se expone en el apartado ?? . A continuación, se inicia la actividad por medio de este objeto **Intent**.

```

1      private void notifyAlertToNotification(String cad){
2          NotificationCompat.Builder mBuilder = (NotificationCompat.Builder) new
3              NotificationCompat.Builder(this)
4                  .setSmallIcon(R.drawable.ic_notify_icon)
5                  .setContentTitle("DeafAlert")
6                  .setContentText(cad)
7                  .setLights(Color.WHITE, 500, 1000)
8                  .setOnlyAlertOnce(true)
9                  .setPriority(2)
10                 .setVibrate(new long[] { 1000, 500, 500, 500, 500 })
11                 .setAutoCancel(true);

```

```
11     Intent resultIntent = new Intent(this, MainActivity.class);
12
13     TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
14     stackBuilder.addParentStack(MainActivity.class);
15     stackBuilder.addNextIntent(resultIntent);
16     PendingIntent resultPendingIntent = stackBuilder.getPendingIntent(0,
17         PendingIntent.FLAG_UPDATE_CURRENT);
18     mBuilder.setContentIntent(resultPendingIntent);
19     NotificationManager mNotificationManager = (NotificationManager)
20         getSystemService(Context.NOTIFICATION_SERVICE);
21     mNotificationManager.notify(NotifCount, mBuilder.build());
22     NotifCount++;
23 }
```

Finalmente, el último método de la actividad principal, **notifyAlertToNotification**, utilizado en el hilo Runnable explicado anteriormente, es el encargado de generar y enviar notificaciones al sistema. Lo primero que se realiza en esta función es crear un objeto Builder de la clase NotificationCompat, el cual se inicializa con una serie de opciones para la notificación, como son el icono de la notificación, el título, el texto que debe contener, y distintos tipos de alerta como el color con el que el led del smartphone debe parpadear, la vibración, y cómo generar estas alertas. De este modo, la persona que utilice la aplicación podrá sentir mediante vibración o notar visualmente la alerta mediante el led, sin necesidad de tener el dispositivo desbloqueado.

Tras crear este objeto, se crea una instancia de la clase Intent asociada a la actividad principal de la aplicación, y mediante un objeto de la clase TaskStackBuilder, se asigna el camino de vuelta al pulsar el botón “Volver” del propio dispositivo. A continuación, mediante un objeto de la clase NotificationManager se obtiene el servicio del sistema relacionado con las notificaciones, y se notifica la alerta mediante el método **notify** propio de esa clase, utilizando un número de identificación para la notificación, y el objeto builder creado al principio de la función.

10.3.3.2.2. Gestión de alertas

La gestión de las alertas a mostrar en la aplicación, así como la funcionalidad de refrescar las mismas en la lista de alertas dentro de la vista principal, se realiza mediante la clase **AlertListAdapter**, clase que hereda de la clase BaseAdapter, clase java de Android utilizada para actualizar el contenido de una vista ListView, vista utilizada en la vista principal para mostrar la lista de alertas.

```
1 public class AlertListAdapter extends BaseAdapter{
2     private ArrayList<String> alerts;
3     private static LayoutInflater inflater = null;
4
5
6     public AlertListAdapter(Activity activity, ArrayList<String> alerts){
7         this.alerts = alerts;
8         this.inflater = activity.getLayoutInflater();
9     }
10    public class ViewHolder
11    {
12        TextView raspberryText;
13    }
```

MEMORIA

El constructor de la clase recibe un objeto Activity, la actividad asociada al Adapter, y un ArrayList de String, la lista de alertas. Mediante la instancia de Activity obtiene un objeto inflater, necesario para obtener más adelante el objeto ListView a actualizar.

Además, se crea una subclase ViewHolder, la cual incluye un objeto de tipo TextView, tipo utilizado para generar cada elemento de la lista ListView.

```
1 public void setData(String newalert){
2     this.alerts.add(newalert);
3     this.notifyDataSetChanged();
4 }
5
6
7 public void remove(int position){
8     this.alerts.remove(position);
9     this.notifyDataSetChanged();
10 }
```

Los métodos encargados de manipular la lista son **setData** y **remove**. El primero recibe una cadena con la alerta, la almacena en la lista, y mediante el método **notifyDataSetChanged** actualiza la información mostrada en la vista principal, refrescando el ListView asociado. El segundo método elimina un elemento de la lista y luego notifica de igual manera el cambio.

```
1 @Override
2 public View getView(int position, View convertView, ViewGroup parent) {
3     View vi = convertView;
4     ViewHolder holder;
5
6     if(convertView==null){
7         vi = inflater.inflate(R.layout.item,null);
8         holder = new ViewHolder();
9         holder.raspberryText = (TextView)vi.findViewById(R.id.textRaspberry);
10        vi.setTag(holder);
11    }else{
12        holder = (ViewHolder)vi.getTag();
13    }
14    holder.raspberryText.setText(this.alerts.get(position));
15    return vi;
16 }
```

Es necesario además sobrecargar el método **getView** de BaseAdapter, para poder obtener la vista a modificar cuando se deba actualizar el contenido de la lista de alertas. Este método usa un objeto de tipo View y otro de la subclase creada anteriormente, ViewHolder. Primero se obtiene el layout o vista a modificar de la aplicación mediante el inflater obtenido en el constructor, para a partir de esta referencia obtener la vista interna que actualizar, mediante su identificador. Hecho esto, ya se puede incluir el texto de la alerta en el TextView asociado al objeto ViewHolder utilizado.

10.3.3.2.3. Menú de configuración

El menú de configuración está basado en el menú generado por Android Studio. Este menú incluye opciones sobrantes, por lo que se retiran todas las partes sobrantes dejando un único submenú con un campo de texto a rellenar, para configurar la dirección IP del servidor de alertas.

Este menú se implementa mediante la clase `SettingsActivity`, el cual incluye subclases en su interior que heredan de la clase `PreferenceFragment` (propia del SDK de Android) por cada submenú. En este caso, al haber un único submenú, hay una única subclase, **`ServerAlertsPreferenceFragment`**.

```

1  public static class ServerAlertsPreferenceFragment extends PreferenceFragment
2      {
3      private static final Pattern PARTIAL_IP_ADDRESS =
4          Pattern.compile("^((25[0-5]|2[0-4][0-9]|[0-1][0-9]{2}|[1-9][0-9]|[0-9])
5              \\.){0,3}" +
6              "((25[0-5]|2[0-4][0-9]|[0-1][0-9]{2}|[1-9][0-9]|[0-9])){0,1}$");
7
8      private EditTextPreference IEditTextPref;
9
10     @Override
11     public void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         addPreferencesFromResource(R.xml.pref_server_alerts);
14         setHasOptionsMenu(true);
15
16         IEditTextPref = (EditTextPreference) getPreferenceManager().findPreference
17             ("ip_address");
18         IEditTextPref.setOnPreferenceChangeListener(new Preference.
19             OnPreferenceChangeListener() {
20             public boolean onPreferenceChange(Preference preference, Object
21                 newValue) {
22                 try {
23                     return PARTIAL_IP_ADDRESS.matcher(newValue.toString()).matches();
24                 }
25                 catch (Exception e) {
26                     return false;
27                 }
28             }
29         });
30
31         bindPreferenceSummaryToValue(findPreference("ip_address"));
32     }
33 }

```

Esta clase cuenta con una constante de la clase `Pattern`, una plantilla, que define como debe estar formada una dirección IP, para poder comparar la preferencia establecida con esta plantilla, y así su validez.

Es en el método **`onCreate`** donde se genera el campo para establecer la dirección IP. A continuación se genera una vista de texto para preferencias (`EditTextPreference`). Una vez hecho, se obtiene la preferencia con identificador **`ip_address`**, y se establece una escucha para cuando

MEMORIA

el valor de esta preferencia cambie, con el método **setOnPreferenceChangeListener**. Como parámetro de este método, se crea un objeto **onPreferenceChangeListener**, para el cual se redefine la función **onPreferenceChange**.

Esta función será la encargada de quedar a la espera de un cambio en el valor de la preferencia, comprobando cuando esto ocurra, si el valor es una dirección IP válida, mediante el método **matcher** de la clase **Pattern**.

Finalmente, se almacena el valor de la preferencia mediante el método **bindPreferenceSummaryToValue**.

11. Planificación Temporal

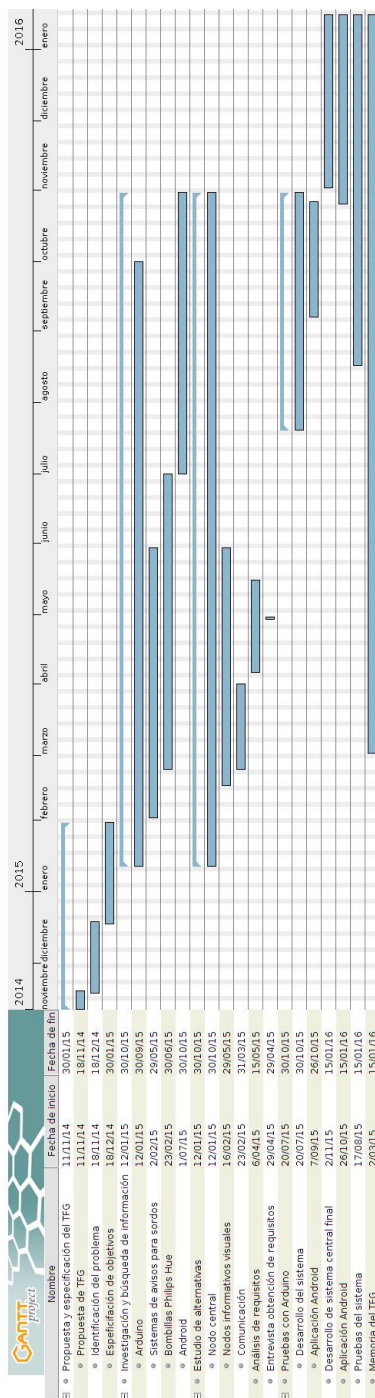


Figura 11.1: Diagrama de Gantt de la planificación temporal del Proyecto

12. Resumen del Presupuesto

Los costes de presupuesto para la implantación en una vivienda real son de 277.08 €, debiéndose sumar además el coste de mano de obra, que asciende a los 130 €. Esto supone un **coste total de 407.08 €**. Se debe tener en cuenta además que este precio es el relativo a instalar 3 bombillas Hue, teniendo un coste de 52.99 € por bombilla extra a instalar [?].

En el apartado ?? se desglosa el presupuesto del Proyecto de forma más detallada.



SISTEMA DE AVISOS PARA DISCAPACITADOS AUDITIVOS

REF: 000001

ANEXOS

- **CLIENTE:** UNIVERSIDAD DE CÁDIZ (ESCUELA SUPERIOR DE INGENIERÍA)
AVENIDA DE LA UNIVERSIDAD DE CÁDIZ N° 10, 11519 PUERTO REAL
956 48 32 00
DIRECCION.ESI@UCA.ES
- **AUTOR:** RAÚL DÍEZ SÁNCHEZ
INGENIERO INFORMÁTICO
77171812-G
RA.DIEZSANCHEZ@ALUM.UCA.ES

Cádiz, 17 de enero de 2016

13. Documentación de Entrada

13.1. Cuestionario realizado a personas con discapacidad auditiva

CUESTIONARIO

El presente cuestionario está enmarcado en un Trabajo de Fin de Grado en el que se pretende crear un sistema de avisos para sordos basado en nuevas tecnologías y con la característica de que sea económico.

1.- ¿Qué avisos son importantes para ti en tu hogar?

- ☐ Llamada a la puerta.
- ☐ Llamada al telefonillo.
- ☐ Llamada al teléfono.
- ☐ Otros avisos (indique cuál).

2.- En la actualidad ¿cómo te percatas de estos avisos?

- ☐ Te avisa un oyente.
- ☐ Luces (explique cómo).
- ☐ Vibración (explique cómo).
- ☐ Otra forma (explique cuál y cómo)

3.- Haciendo uso de las nuevas tecnologías ¿crees que puede mejorarse?, ¿cómo te gustaría percartarte de los avisos?

- ☐ Luces (explique cómo).
- ☐ Vibración (explique cómo).
- ☐ Otra forma (explique cuál y cómo).

4.- Continuando con las nuevas tecnologías ¿crees que sería útil recibir las mismas alertas en smartphones Android?

- ☐ Sí, utilizo un smartphone y sería útil.
- ☐ No, no utilizo smartphone o no me parece útil.

5.- Comentarios:

14. Análisis y Diseño del Sistema

14.1. Diseño de arquitectura del Sistema

El diseño final de la arquitectura del sistema viene determinado por la figura ??.

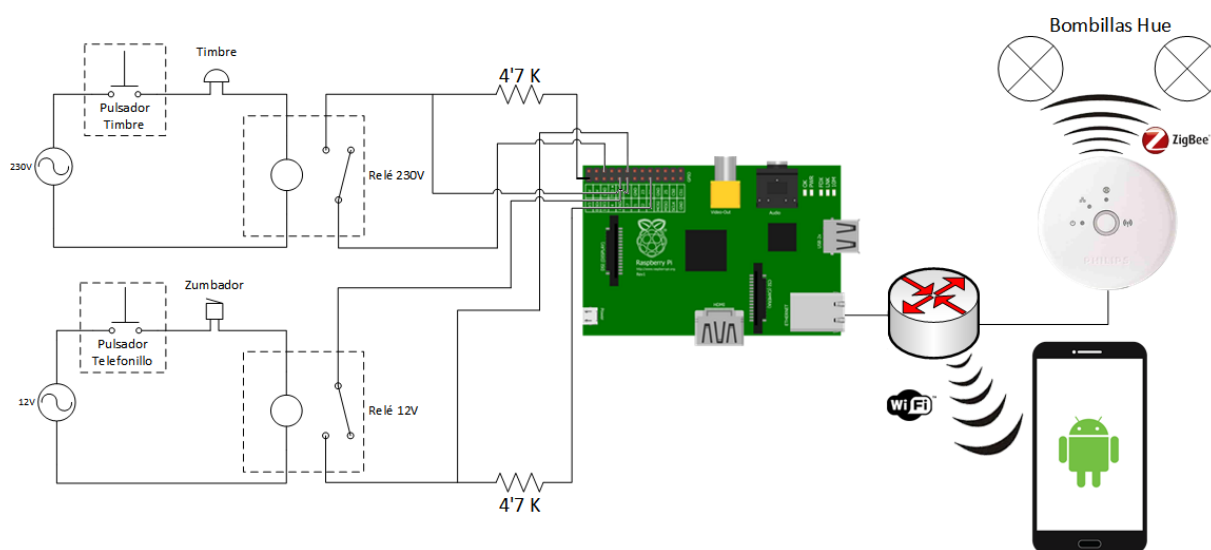


Figura 14.1: Diseño final de la arquitectura del sistema.

14.2. Diseño de sistemas de interconexión con nodo central

Los sistemas de interconexión del telefonillo y el timbre con el nodo central son implementados por dos placas de circuito impreso, una para cada conexión.

14.2.1. Placa de circuito impreso para conexión con telefonillo

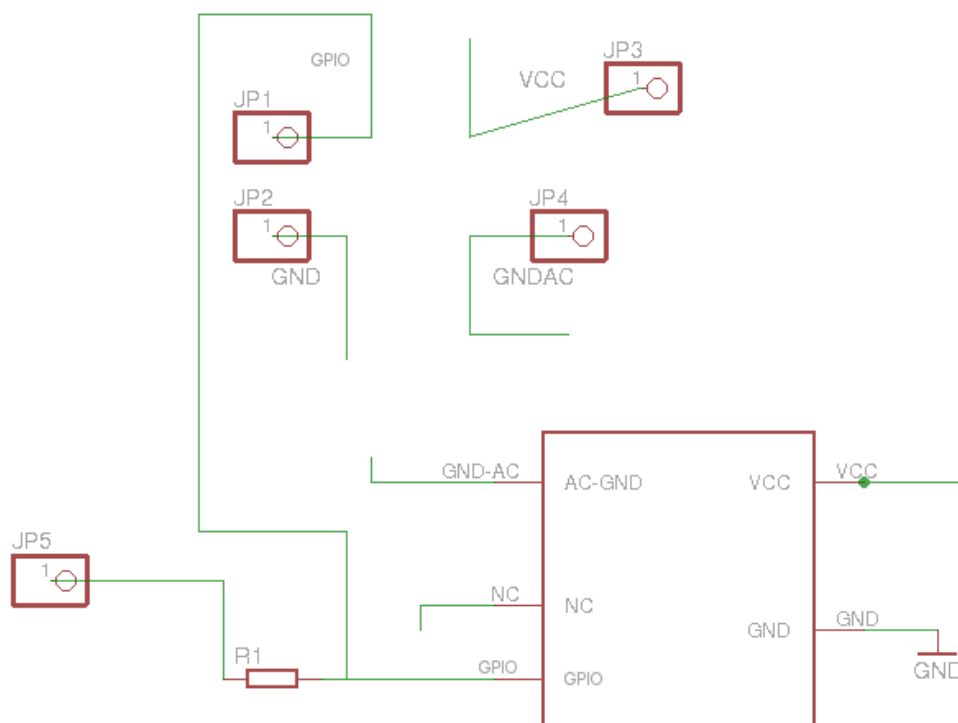


Figura 14.2: Esquemático de placa de circuito impreso para conexión con telefonillo.

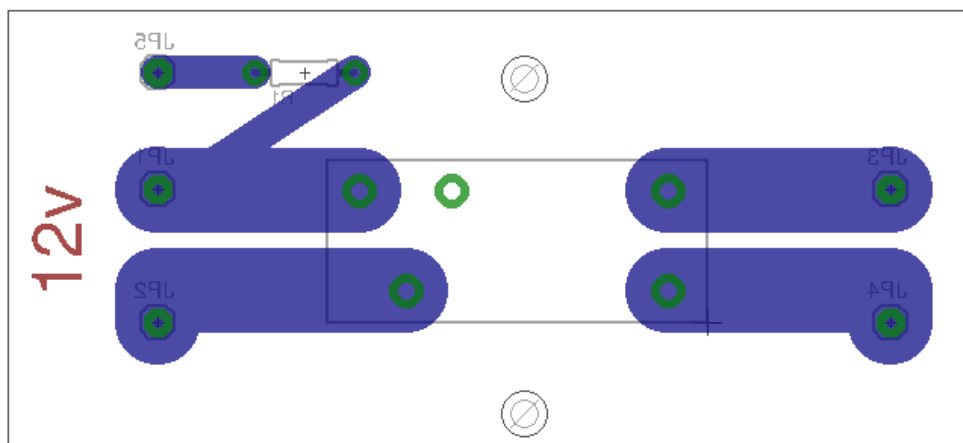


Figura 14.3: Diseño de placa de circuito impreso para conexión con telefonillo.

14.2.2. Placa de circuito impreso para conexión con timbre

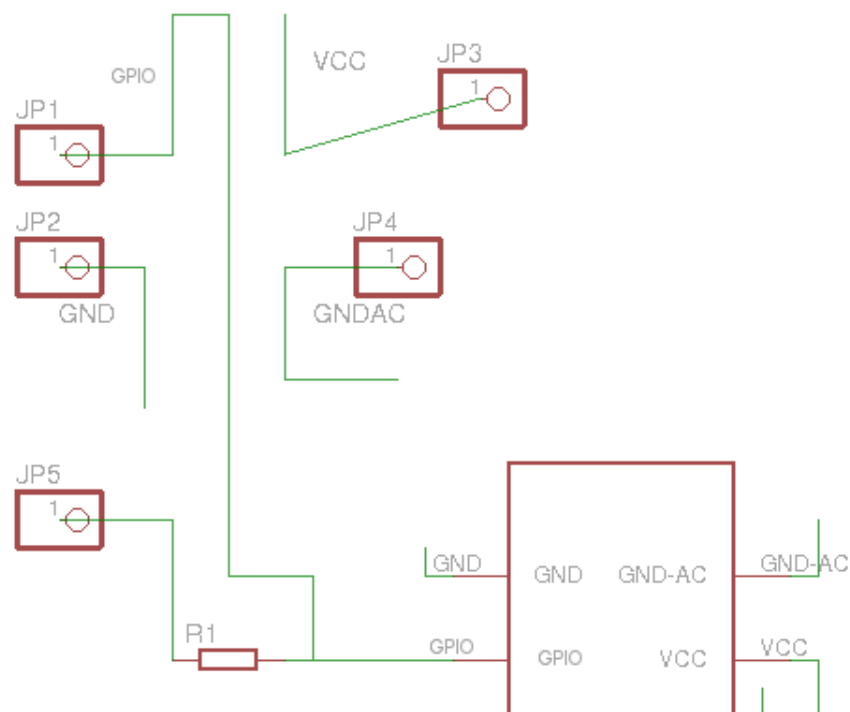


Figura 14.4: Esquemático de placa de circuito impreso para conexión con timbre.

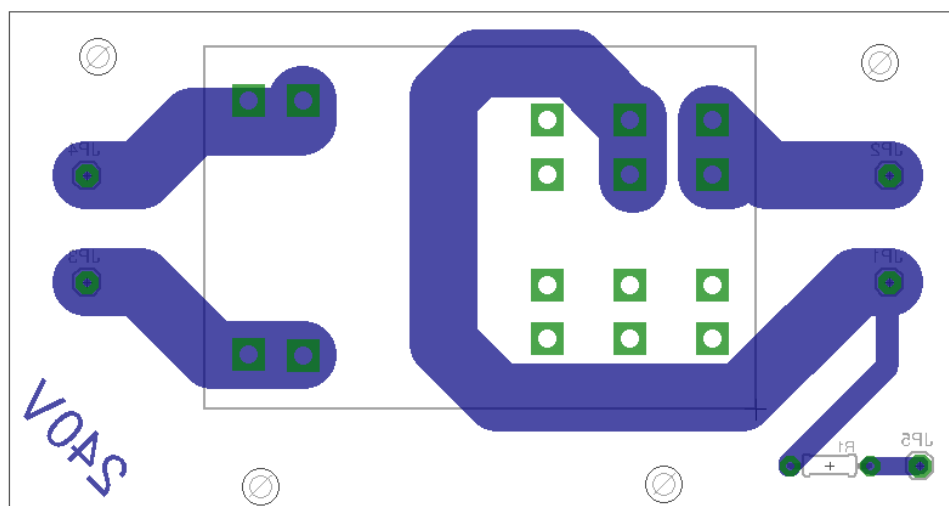

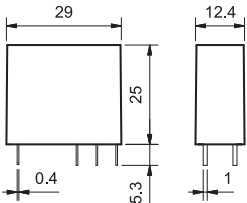

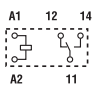
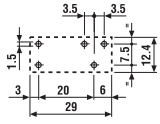

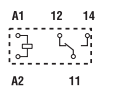
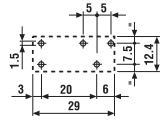

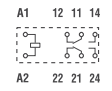
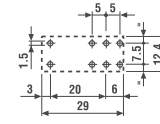













Figura 14.5: Diseño de placa de circuito impreso para conexión con timbre.

14.3. Hojas técnicas de componentes empleados

14.3.1. Hoja técnica de relé 40.31 de 12V

 Serie 40 - Mini-relé para circuito impreso enchufable 8 - 10 - 16 A			
Características Relé con 1 o 2 contactos 40.31 - 1 contacto 10 A (pas 3.5 mm) 40.51 - 1 contacto 10 A (pas 5 mm) 40.52 - 2 contactos 8 A (pas 5 mm) Montaje en circuito impreso - directo o en zócalo Montaje en carril de 35 mm (EN 60715) - en zócalos con bornes a pletina o de conexión rápida <ul style="list-style-type: none"> Bobina DC (estándar o sensible) y bobina AC Contactos sin Cadmio 8 mm, 6 kV (1.2/50 µs) entre bobina y contactos UL Listing (combinaciones relé/zócalo) Estando al flux: RT II estándar, (disponible en versión RT III) Zócalos serie 95 Modulos de señalización y protección CEM Modulos temporizados serie B6  <p>PARA CARGAS DE MOTORES Y "PILOT DUTY" HOMOLOGADAS POR UL VER "Información Técnica General" página V</p>	40.31  <ul style="list-style-type: none"> Reticulado 3.5 mm 1 contacto 10 A Montaje en circuito impreso o en zócalo serie 95   <p>Vista parte inferior</p>	40.51  <ul style="list-style-type: none"> Reticulado 5 mm 1 contacto 10 A Montaje en circuito impreso o en zócalo serie 95   <p>Vista parte inferior</p>	40.52  <ul style="list-style-type: none"> Reticulado 5 mm 2 contactos 8 A Montaje en circuito impreso o en zócalo serie 95   <p>Vista parte inferior</p>
Características de los contactos			
Configuración de contactos	1 contacto conmutado	1 contacto conmutado	2 contactos conmutados
Corriente nominal/Máx. corriente instantánea A	10/20	10/20	8/15
Tensión nominal/Máx. tensión de conmutación V AC	250/400	250/400	250/400
Carga nominal en AC1 VA	2500	2500	2000
Carga nominal en AC15 (230 V AC) VA	500	500	400
Motor monofásico (230 V AC) kW	0.37	0.37	0.3
Capacidad de ruptura en DC1: 30/110/220 VA	10/0.3/0.12	10/0.3/0.12	8/0.3/0.12
Carga mínima conmutable mW (V/mA)	300 (5/5)	300 (5/5)	300 (5/5)
Material estándar de los contactos	AgNi	AgNi	AgNi
Características de la bobina			
Tensión nominal V AC (50/60 Hz)	6 - 12 - 24 - 48 - 60 - 110 - 120 - 230 - 240		
de alimentación (U _N) V DC	5 - 6 - 7 - 9 - 12 - 14 - 18 - 21 - 24 - 28 - 36 - 48 - 60 - 90 - 110 - 125		
Potencia nominal en AC/DC/DC sens. VA (50 Hz)/W/W	1.2/0.65/0.5	1.2/0.65/0.5	1.2/0.65/0.5
Campo de funcionamiento AC	(0.8...1.1)U _N	(0.8...1.1)U _N	(0.8...1.1)U _N
DC/DC sensible	(0.73...1.5)U _N /(0.73...1.75)U _N	(0.73...1.5)U _N /(0.73...1.75)U _N	(0.73...1.5)U _N /(0.73...1.75)U _N
Tensión de mantenimiento AC/DC	0.8 U _N / 0.4 U _N	0.8 U _N / 0.4 U _N	0.8 U _N / 0.4 U _N
Tensión de desconexión AC/DC	0.2 U _N / 0.1 U _N	0.2 U _N / 0.1 U _N	0.2 U _N / 0.1 U _N
Características generales			
Vida útil mecánica AC/DC ciclos	10 · 10 ⁶ / 20 · 10 ⁶	10 · 10 ⁶ / 20 · 10 ⁶	10 · 10 ⁶ / 20 · 10 ⁶
Vida útil eléctrica con carga nominal AC1 ciclos	200 · 10 ³	200 · 10 ³	100 · 10 ³
Tiempo de respuesta: conexión/desconexión ms	7/3 - (12/4 sensible)	7/3 - (12/4 sensible)	7/3 - (12/4 sensible)
Aislamiento entre bobina y contactos (1.2/50 µs) kV	6 (8 mm)	6 (8 mm)	6 (8 mm)
Rigidez dieléctrica entre contactos abiertos V AC	1000	1000	1000
Temperatura ambiente °C	-40...+85	-40...+85	-40...+85
Categoría de protección	RT II**	RT II**	RT II**
Homologaciones (según los tipos)	          		

** Ver información técnica "Indicaciones sobre los procedimientos de soldadura automatica" página II.

14.3.2. Hoja técnica de relé T92S7A12 de 240V



General Purpose
High Power PCB Relays

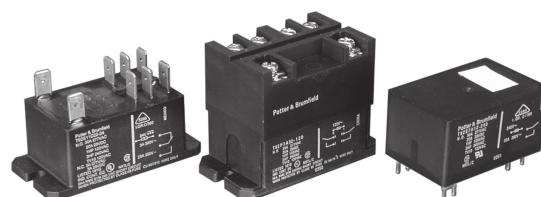
Potter & Brumfield

T92 Series Two-pole 30A PCB or Panel Mount Relay

- 40A, 2 form A (NO) and 2 form C (CO) switching capability
- Designed to control compressor loads to 3.5 tons, 110LRA / 25.3FLA
- Meets requirements of UL 508 and UL 873 spacings - 8mm through air, 9.5mm over surface
- Meets requirements of VDE 8mm spacing, 4kV dielectric coil-to-contact
- Meets requirements of UL Class F construction
- UL approved for 600VAC switching (1.5HP)
- New screw terminal version (consult factory for availability, ratings)

Typical applications

HVAC, residential / commercial appliances, industrial controls.



Approvals

UL E58304 (Recognized and Listed); CSA LR48471; VDE 40019600

Technical data of approved types on request.

Contact Data

Contact arrangement	2 form A (NO), 2 form C (CO)
Rated voltage	277VAC
Max. switching voltage	600VAC
Rated current	30A NO; 3A NC
Limiting continuous current	40A NO; 3A NC
Limiting making current	40A NO; 3A NC
Limiting breaking current	40A NO; 3A NC
Contact material	AgSnOInO, AgCdO
Min. recommended contact load	500ma (NO) / 100ma (NC), 12VAC
Frequency of operation, with load	360hr
Operate/release time max., including bounce	25/25ms

Contact ratings ¹⁾

Type	Load	Cycles
UL508		
AgCdO		
NO	40A, 277VAC, resistive	6x10 ³
NO	30A, 120/277VAC, resistive	100x10 ³
NO	10A, 600VAC, general purpose	100x10 ³
NO	1HP, 120VAC	100x10 ³
NO	3HP, 240VAC	1x10 ³
NO	1.5HP, 480 or 600VAC	100x10 ³
NO	110LRA/25.3FLA, 240VAC (DC coil only)	100x10 ³
NO	60LRA/14FLA, 240VAC (AC coil only)	100x10 ³
NO	3A, 240VAC, pilot duty	100x10 ³
NO	20A, 28VDC, resistive	100x10 ³
NO	TV10, 120VAC	100x10 ³
NC	3A, 277VAC	100x10 ³
NC	2A, 480VAC	100x10 ³
NC	1A, 600VAC	100x10 ³
AgSnOInO		
NO	30A, 120/277VAC, resistive (DC coil only)	200x10 ³
NO	30A, 120/277VAC, resistive (AC coil only)	100x10 ³
NO	20A, 480VAC, resistive	100x10 ³
NO	1.5HP, 120VAC, 2 pole making/breaking (Fig.1)	100x10 ³
NO	3HP, 240VAC, 3 phase (DC coil only)	100x10 ³
NO	3HP, 480VAC, 3 phase (DC coil only)	100x10 ³
NO	2HP, 600VAC, 3 phase (DC coil only)	100x10 ³
VDE		
AgCdO, flange mount relays		
NO	20A, 400VAC	100x10 ³
NC	3A, 400VAC	30x10 ³
CO	20A NO / 3A NC, 400VAC	30x10 ³
AgCdO, PC mount relays		
NO	30A, 400VAC	100x10 ³
NC	3A, 400VAC	30x10 ³
CO	30A NO / 3A NC, 400VAC	30x10 ³

Contact ratings ¹⁾ (continued)

ARI 780-86 Endurance Test (section 6.6):

HVAC Definite Purpose Contactor Standard

Normally Open Contacts

Single Phase/Two Pole (Both poles together switching a single load)

110 LRA, 25.3 FLA, 200K operations (DC Coil)

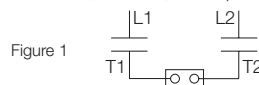


Figure 1

Single Phase Per Pole (Single load per pole)

110 LRA, 18 FLA, 200K operations (DC Coil).

60 LRA, 14 FLA, 200K operations (AC Coil).

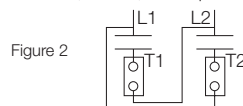


Figure 2

¹⁾ Contact ratings at 25°C (unless otherwise noted) with relay properly vented.
FLA, LRA ratings are compatible with 3.5 ton compressor applications.

Mechanical endurance	10x10 ⁶ ops.
----------------------	-------------------------

Coil Data

Coil voltage range	5 to 110VDC; 12 to 240VAC
Max. coil power	1.7W; 4.0VA
Max. coil temperature	155°C
Coil insulation system according UL	Class F

Coil versions, DC coil

Coil code	Rated voltage VDC	Operate voltage VDC	Release voltage VDC	Coil resistance Ω±10%	Rated coil power W
6	6	4.5	0.6	22	1.7
9	9	6.75	0.9	48	1.7
12	12	9	1.2	86	1.7
18	18	13.5	1.8	197	1.7
24	24	18	2.4	350	1.7
48	48	36	4.8	1390	1.7
110	110	82.5	11	7255	1.7

Coil versions, AC coil

Coil code	Rated voltage VAC	Frequency Hz	Operate voltage VAC, 60Hz	Release voltage VAC, 60Hz	Coil resistance Ω±10%	Rated coil power VA
12	12	60	9.6	1.2	9.1	4
24	24	60	19.2	2.4	36.6	4
120	110/120	50/60	96	12	950	4
240	220/240	50/60	192	24	3800	4
277	250/277	50/60	222	28	5485	4

All figures are given for coil without preenergization, at ambient temperature +23°C.

14.4. Diseño de prototipo de aplicación para smartphone

El diseño del prototipo de la aplicación se compone de una pantalla principal con la lista de alertas y un botón para menú de configuración (figuras ?? y ??), un sistema de borrado de alertas mediante ventanas emergentes (figura ??), y un menú de configuración (figura ??) en el que se pueda introducir la dirección del servidor de alertas, el nodo central del sistema.



Figura 14.6: Prototipo de la pantalla principal de la aplicación para smartphones.



Figura 14.7: Prototipo del menú desplegable.

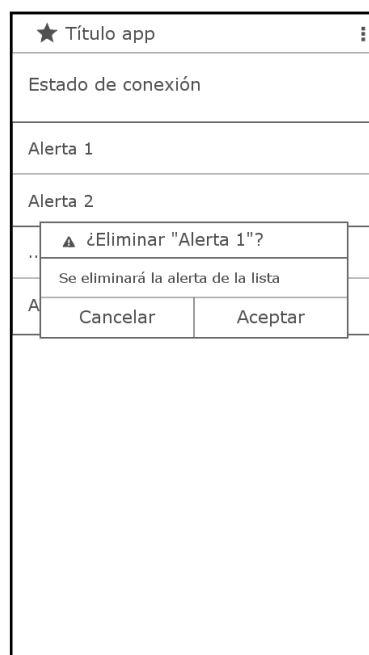
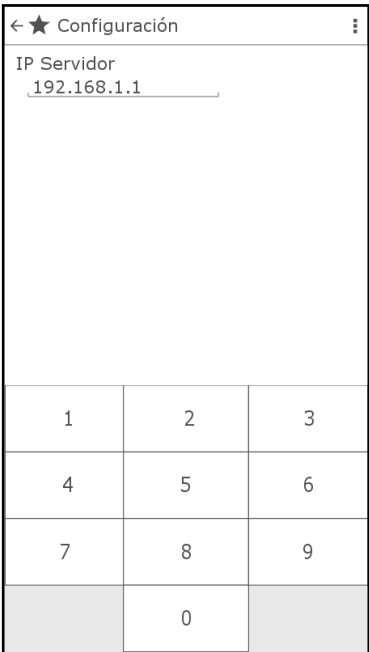


Figura 14.8: Prototipo de ventana emergente para eliminar alertas.



← ★ Configuración

IP Servidor
192.168.1.1

1	2	3
4	5	6
7	8	9
	0	

The image shows a mobile application configuration screen. At the top, there is a header bar with a back arrow, a star icon, and the text 'Configuración'. Below the header, there is a section for 'IP Servidor' with a text input field containing '192.168.1.1'. At the bottom of the screen, there is a numeric keypad with a 3x3 grid of buttons. The buttons are labeled with numbers 1 through 9, and a 0 button in the center of the bottom row. The first and last buttons of the bottom row are shaded gray.

Figura 14.9: Prototipo de menú de configuración.

15. Instalación de Sistema Operativo en Raspberry Pi

Para obtener un funcionamiento igual al de las pruebas realizadas, se especifica en este apartado la instalación y configuración del Sistema Operativo utilizado en Raspberry Pi, de cara a la implantación del Proyecto en un entorno final.

15.1. Sistema Operativo base

Se elige Arch Linux ARM como Sistema Operativo base para Raspberry Pi. Para su instalación, es necesario introducir una tarjeta SD (o microSD con adaptador SD) en un ordenador con algún sistema GNU/Linux instalado y conexión a internet. A continuación, ingresar a un terminal e introducir los siguientes comandos, como superusuario o usuario **root**:

1. Iniciar fdisk en la SD (donde **mmcblk0** es el identificador de la tarjeta SD):

```
c+c1# fdisk /dev/mmcblk0
```

2. Dentro del prompt de fdisk, borrar las particiones antiguas y crear nuevas:

- a) Escribir **o** para borrar las particiones de la SD.
- b) Escribir **n** y luego **p** para crear partición primaria. Introducir **1** para crear la primera partición, pulsar **ENTER** y luego escribir **+100M** para los sectores a comprender.
- c) Escribir **t** y luego **c** para formatear la partición como FAT32.
- d) Escribir **n** y luego presionar **ENTER** 4 veces para crear la segunda partición primaria con los valores por defecto
- e) Por último, escribir **w** para escribir los cambios a la tarjeta SD y salir.

3. Crear y montar el sistema FAT32:

```
c+c1# mkfs.vfat /dev/mmcblk0p1 && mkdir boot && mount  
→ /dev/mmcblk0p1 boot
```

4. Crear y montar el sistema ext4:

```
c+c1# mkfs.ext4 /dev/mmcblk0p2 && mkdir root && mount  
→ /dev/mmcblk0p2 root
```

5. Descargar y descomprimir Arch Linux ARM a la tarjeta SD:

```
c+c1# wget  
→ http://archlinuxarm.org/os/ArchLinuxARM-rpi-latest.tar.gz &&  
→ bsdtar -xpf ArchLinuxARM-rpi-latest.tar.gz -C root && sync
```

6. Mover archivos necesarios a la partición boot:

```
c+c1# mv root/boot/* boot
```

7. Desmontar particiones:

```
c+c1# umount boot root
```

Una vez acabado esto, Arch Linux ARM estará instalado en la tarjeta SD.

15.2. Configuración del Sistema

Con el Sistema Operativo ya instalado, se debe introducir la tarjeta SD en la Raspberry Pi, conectando además un cable Ethernet y el cable de corriente. Una vez encendida, ingresar mediante un terminal vía SSH desde un ordenador a la Raspberry Pi, utilizando **alarm** como usuario y **alarm** como contraseña. Para configurar Arch Linux ARM e instalar los programas necesarios para el correcto funcionamiento del sistema de alertas, ejecutar las siguientes órdenes:

1. Ingresar como superusuario, utilizando **root** como contraseña:

```
c+c1# su
```

2. Configurar los locales del sistema a Español:

```
c+c1# sed -i "s^#es_ES.UTF-8 UTF-8^es_ES.UTF-8 UTF-8^g"
↪ /etc/locale.gen
c+c1# locale-gen
c+c1# localectl set-locale LANG="es_ES.UTF8",
↪ LC_TIME="es_ES.UTF8"
c+c1# rm -f /etc/localtime ; ln -s
↪ /usr/share/zoneinfo/Europe/Madrid /etc/localtime
```

3. Sincronizar los repositorios y actualizar el sistema:

```
c+c1# pacman -Syu --noconfirm
```

4. Instalación de paquetería necesaria para desarrollo, así como las herramientas git, Python y Nmap:

```
c+c1# pacman -S base-devel wget python2 git bash-completion
↪ python2-pip nmap --noconfirm
```

5. Añadir el grupo **wheel** a los usuarios de **sudo**, grupo al que pertenece el usuario **alarm**:

```
c+c1# sed -i "s^# %wheel ALL=(ALL) ALL^%wheel ALL=(ALL)
↪ ALL^g" /etc/sudoers
```

6. Reiniciar el sistema para aplicar todos los cambios:

```
c+c1# reboot
```

7. Una vez iniciado de nuevo e ingresado mediante el usuario anterior (**alarm**), instalar la librería RPi.GPIO, necesaria para el uso de los pines GPIO de Raspberry Pi:

```
c+c1# sudo pip2 install RPi.GPIO
```

Una vez terminado, el sistema ya está listo para introducir en él el programa del sistema de alertas y ponerlo en marcha.

16. Estimación de Tamaño y Esfuerzos

Para la implantación de este sistema, es necesario contar con los siguientes componentes:

- Una Raspberry Pi, junto con su adaptador de corriente, tarjeta SD, y latiguillo Ethernet RJ-45 o adaptador WiFi USB.
- Un puente Philips Hue, junto con su adaptador de corriente y latiguillo Ethernet RJ-45.
- Tantas bombillas Philips Hue de rosca E27 como se deban implantar en la vivienda. En este caso, como no se implanta el sistema en una vivienda final, se han utilizado dos bombillas.
- Un smartphone Android, con la aplicación final instalada.
- 2 placas de circuito impreso (PCB) diseñadas conforme a los diseños incluidos en estos Anexos, junto con la electrónica y componentes necesarios para su montaje, que son:
 - Un relé 4031 de 12V a 5V.
 - Un relé T92S7A12 de 240V a 5V.
 - 4 terminales para PCB de doble conexión.
 - 2 resistencias de 4,7 K Ω .
 - 2 conectores macho para PCB.
- Un router WiFi o cable/módem WiFi.



SISTEMA DE AVISOS PARA DISCAPACITADOS AUDITIVOS

REF: 000001

ESPEFICICACIONES DEL SISTEMA

- **CLIENTE:** UNIVERSIDAD DE CÁDIZ (ESCUELA SUPERIOR DE INGENIERÍA)
AVENIDA DE LA UNIVERSIDAD DE CÁDIZ Nº 10, 11519 PUERTO REAL
956 48 32 00
DIRECCION.ESI@UCA.ES
- **AUTOR:** RAÚL DÍEZ SÁNCHEZ
INGENIERO INFORMÁTICO
77171812-G
RA.DIEZSANCHEZ@ALUM.UCA.ES

Cádiz, 17 de enero de 2016

17. Especificaciones del Sistema

17.1. Estudio de las especificaciones

17.1.1. Estudio de dominio del problema

El dominio de este problema engloba a todas las personas con discapacidad auditiva que puedan requerir el uso del sistema a implementar. Para tratar el problema en cuestión, se realiza una entrevista con un grupo de personas con esta discapacidad.

17.1.2. Selección de personas a entrevistar

Para la obtención de unos datos ajustados a la realidad, se entrevista a personas con discapacidad auditiva pertenecientes a la asociación “Albor Cádiz”, de la que hay disponible más información en su página de Facebook: <https://www.facebook.com/alborcadiz>.

17.1.3. Objetivo y contenido de la entrevista

El objetivo de esta entrevista es el de obtener los requisitos del sistema, de la manera más real posible. El contenido de la entrevista se puede encontrar en el apartado ?? del Anexo.

17.1.4. Planificación de la entrevista

La entrevista se realiza el día 29 de Abril de 2015, a las 18:30 horas, en la sede de la asociación “Albor Cádiz”.

17.1.5. Análisis de los resultados obtenidos en la entrevista

1 ¿Qué avisos son importantes para ti en tu hogar?

En la figura ?? se puede observar como las alertas de mayor interés son las llamadas a la puerta (timbre) o al telefonillo, además de varias sugerencias para otros avisos. Queda así descartado el control de alertas para llamadas de teléfono, pues no supone gran interés.

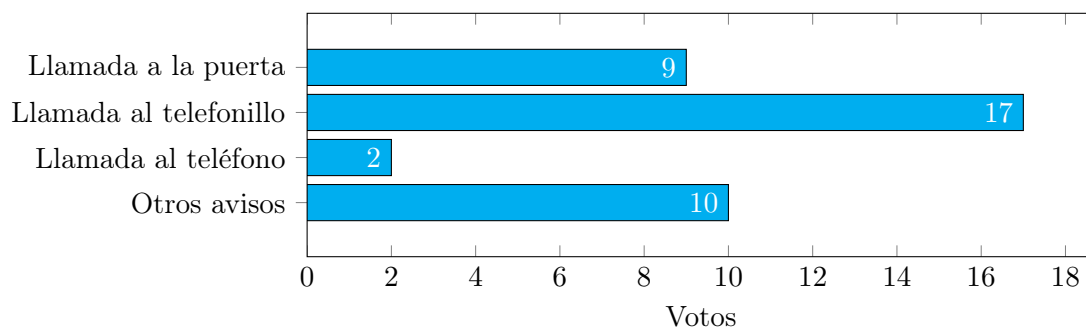


Figura 17.1: Resultados para “¿Qué avisos son importantes para ti en tu hogar?”

2 En la actualidad, ¿cómo te percatas de estos avisos?

Se observa en la figura ?? los resultados para esta pregunta. La mayoría de los encuestados utilizan actualmente sistemas de alertas mediante luces.

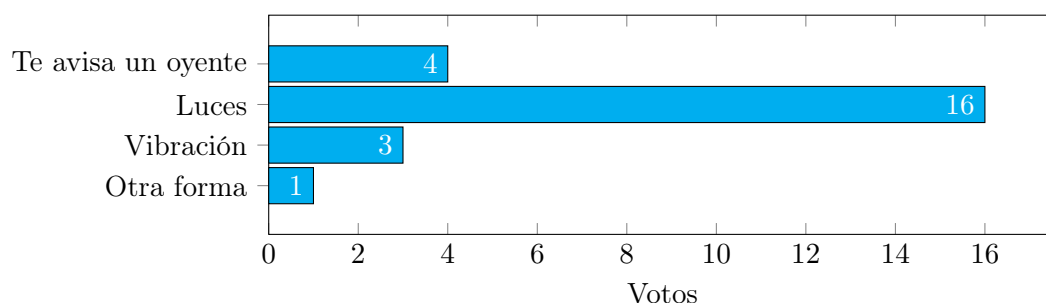


Figura 17.2: Resultados para “En la actualidad, ¿cómo te percatas de estos avisos?”

3 Haciendo uso de las nuevas tecnologías, ¿crees que puede mejorarse?, ¿cómo te gustaría percartarte de los avisos?

Los resultados para esta pregunta se exponen en la figura ?. La mayoría de los encuestados prefieren alertas con luces, en concreto, de colores y parpadeantes, según comentan; además de vibraciones, esta última forma genera cierto rechazo ya que puede sobresaltar al usuario.

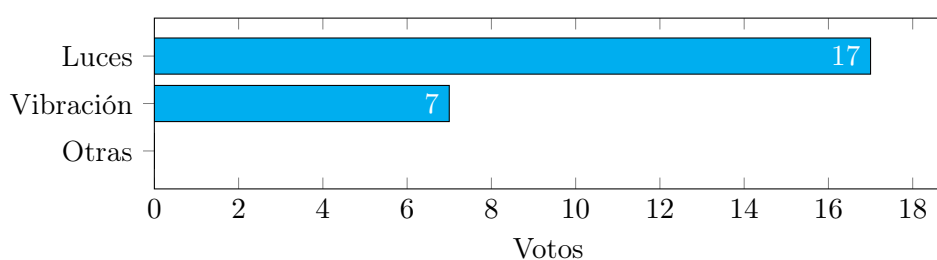


Figura 17.3: Resultados para “Haciendo uso de las nuevas tecnologías, ¿crees que puede mejorarse?, ¿cómo te gustaría percartarte de los avisos?”

4 Continuando con las nuevas tecnologías, ¿crees que sería útil recibir las mismas alertas en smartphones Android?

Para esta última pregunta, en la figura ?? se observan los resultados obtenidos. Casi todos los discapacitados encuestados prefieren recibir las mismas alertas en sus smartphones Android, apostando por el uso de aplicaciones para estas alertas.

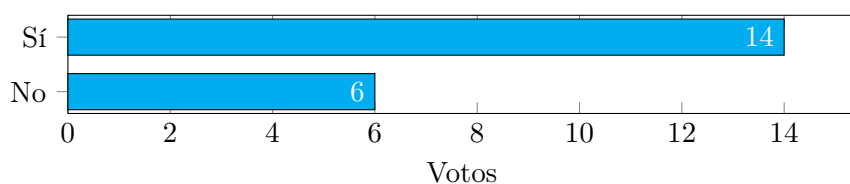


Figura 17.4: Resultados para “Continuando con las nuevas tecnologías, ¿crees que sería útil recibir las mismas alertas en smartphones Android?”

En cualquier caso, los resultados no están lejos de los esperados, llegando así en resumen a las siguientes conclusiones:

- Los avisos tratados serán las llamadas a la puerta de la vivienda (timbre) y las llamadas al telefonillo o portero automático.
- Los avisos se tratarán como alertas visuales mediante luces de colores.
- Además de las luces, se desarrollará una aplicación para Android en la que se notifiquen las mismas alertas, ya que es el sistema operativo para smartphone más extendido en el mercado.

17.2. Descripción del problema

- El usuario tiene una discapacidad auditiva.
- El usuario necesita percatarse de las alertas sonoras que se dan en el entorno de alguna manera.
- Las alertas serán: llamada al telefonillo y llamada al timbre de la puerta de la vivienda.
- Las alertas se deben conectar a un sistema.
- El sistema alertará visualmente al usuario mediante luces de colores, y mostrando notificaciones en su smartphone.

17.3. Objetivos del sistema

OBJ-01	Alertar a usuario.
Descripción	El sistema deberá alertar visualmente mediante el cambio de color de las luces y mediante notificación vía smartphone de las alertas (llamada a la puerta o al telefonillo) al usuario.
Importancia	Alta.
Estabilidad	Alta.
Comentarios	Ninguno.

Cuadro 17.1: Objetivos del sistema.

17.4. Requisitos de información

CRQ-01	Relación entre alertas.
Objetivos asociados	<ul style="list-style-type: none"> ■ OBJ-01 Alertar a usuario.
Requisitos asociados	<ul style="list-style-type: none"> ■ IRQ-01 Información de alertas.
Descripción	Solo se podrá mostrar una alerta a la vez.
Importancia	Alta.
Estabilidad	Alta.

Cuadro 17.2: Requisitos de restricciones de información.

IRQ-01	Información de alertas.
Objetivos asociados	<ul style="list-style-type: none"> ■ OBJ-01 Alertar a usuario.
Requisitos asociados	<ul style="list-style-type: none"> ■ UC-01 Alertar timbre. ■ UC-02 Alertar telefonillo. ■ UC-03 Mostrar alerta.
Descripción	El sistema deberá alertar de los eventos que sucedan. Estos eventos se almacenarán de forma temporal en la memoria del sistema central, a modo de variables del programa.
Datos específicos	<ul style="list-style-type: none"> ■ Fecha y hora. ■ Tipo de alerta (llamada a la puerta o al telefonillo).
Importancia	Alta.
Estabilidad	Alta.

Cuadro 17.3: Requisitos de almacenamiento de información.

17.5. Diagrama de Casos de Uso

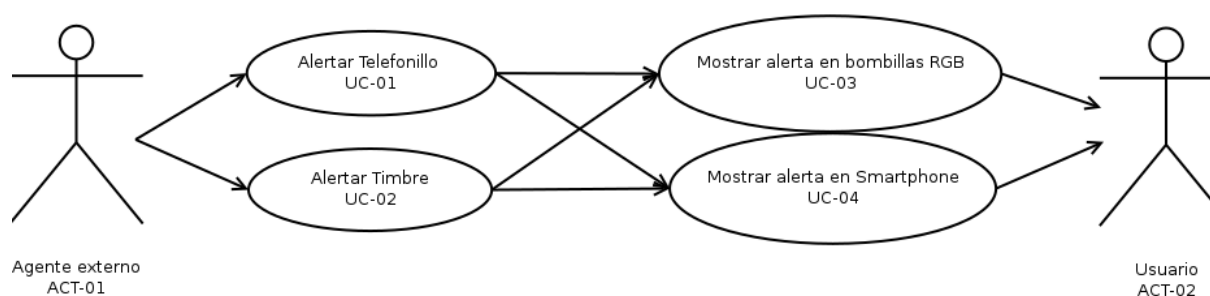


Figura 17.5: Diagrama de Casos de Uso.

17.6. Descripción de actores

ACT-01	Agente externo.
Descripción	Persona externa al sistema que efectúa la alerta.

Cuadro 17.4: Actor 01: agente externo.

ACT-02	Usuario.
Descripción	Usuario final del sistema.

Cuadro 17.5: Actor 02: usuario del sistema.

17.7. Requisitos funcionales (Casos de Uso)

UC-01	Alertar telefonillo.	
Objetivos asociados	■ OBJ-01 Alertar a usuario.	
Requisitos asociados	■ IRQ-01 Información de alertas.	
Descripción	El sistema deberá comportarse tal como se describe en este Caso de Uso cuando un Agente Externo llame al telefonillo.	
Precondición	Un Agente Externo llama al telefonillo.	
Secuencia normal	Paso	Acción
	1	Llaman al telefonillo.
	2	Se envía una alerta al sistema.
	1	Se realizan los Casos de Uso UC-03 y UC-04.
Postcondición	Se envía la alerta al sistema.	
Excepciones	Ninguna.	
Rendimiento	Paso	Cota de tiempo
	2	<1 segundo.
Comentarios	La frecuencia dependerá de factores externos al sistema que se estudia en el Proyecto.	

Cuadro 17.6: Caso de Uso 01: Alertar telefonillo.

UC-02	Alertar timbre.	
Objetivos asociados	■ OBJ-01 Alertar a usuario.	
Requisitos asociados	■ IRQ-01 Información de alertas.	
Descripción	El sistema deberá comportarse tal como se describe en este Caso de Uso cuando un Agente Externo llame al timbre de la puerta.	
Precondición	Un Agente Externo llama al timbre.	
Secuencia normal	Paso	Acción
	1	Llaman al timbre.
	2	Se envía una alerta al sistema.
	1	Se realizan los Casos de Uso UC-03 y UC-04.
Postcondición	Se envía la alerta al sistema.	
Excepciones	Ninguna.	
Rendimiento	Paso	Cota de tiempo
	2	<1 segundo.
Comentarios	La frecuencia dependerá de factores externos al sistema que se estudia en el Proyecto.	

Cuadro 17.7: Caso de Uso 02: Alertar timbre.

UC-03	Mostrar alerta en bombilla RGB.	
Objetivos asociados	■ OBJ-01 Alertar a usuario.	
Requisitos asociados	■ IRQ-01 Información de alertas.	
Descripción	El sistema cambiará el estado de las bombillas RGB de acuerdo a la alerta recibida siempre que estas no ocurran de manera simultánea.	
Precondición	Se dan uno de los dos Casos de Uso asociados a alertas: UC-01 o UC-02.	
Secuencia normal	Paso	Acción
	1	Se recibe la alerta en el sistema.
	2	Se almacena el estado de las bombillas RGB.
	3	Se genera alerta luminosa cambiando el estado de las bombillas RGB.
	4	Se regresa al estado previo a la alerta.
Postcondición	El Usuario percibe la alerta.	
Excepciones	Paso	Acción
	1	Se solapen 2 o más alertas y se omita el aviso.
Rendimiento	Paso	Cota de tiempo
	2	~1 segundo.
Comentarios	La frecuencia dependerá de factores externos al sistema que se estudia en el Proyecto.	

Cuadro 17.8: Caso de Uso 03: Mostrar alerta en bombilla RGB.

UC-04	Mostrar alerta en smartphone.	
Objetivos asociados	■ OBJ-01 Alertar a usuario.	
Requisitos asociados	■ IRQ-01 Información de alertas.	
Descripción	El sistema enviará una notificación al smartphone de acuerdo a la alerta recibida siempre que estas no ocurran de manera simultánea.	
Precondición	Se dan uno de los dos Casos de Uso asociados a alertas: UC-01 o UC-02.	
Secuencia normal	Paso	Acción
	1	Se recibe la alerta en el sistema.
	2	Se envía la notificación al smartphone.
Postcondición	El Usuario percibe la alerta.	
Excepciones	Paso	Acción
	1	Se solapan 2 o más alertas y se omite el aviso.
Rendimiento	Paso	Cota de tiempo
	2	~1 segundo.
Comentarios	La frecuencia dependerá de factores externos al sistema que se estudia en el Proyecto.	

Cuadro 17.9: Caso de Uso 04: Mostrar alerta en smartphone.

17.8. Requisitos no funcionales

Teniendo en cuenta las restricciones obtenidas a través de las encuestas realizadas, los requisitos no funcionales resultantes son los siguientes:

NFR-01: Se deben utilizar bombillas de luz LED RGB, para generar distintos colores en las alertas.

NFR-02: Se debe desarrollar una aplicación para Android en la que se reflejen también las mismas alertas.

Además, según el desarrollo del Proyecto y el estudio de distintas alternativas para la solución final, surgen otros requisitos funcionales:

NFR-03: Las bombillas a utilizar deben ser de rosca E27 (de acuerdo al Informe de diagnóstico del apartado ??).

NFR-04: El nodo central debe ser capaz de manipular peticiones HTTP (de acuerdo a la elección de bombillas en el apartado ??).

NFR-05: El nodo central debe ser capaz de manipular cadenas JSON (de acuerdo a la elección de bombillas en el apartado ??).



SISTEMA DE AVISOS PARA DISCAPACITADOS AUDITIVOS

REF: 000001

PRESUPUESTO

- **CLIENTE:** UNIVERSIDAD DE CÁDIZ (ESCUELA SUPERIOR DE INGENIERÍA)
AVENIDA DE LA UNIVERSIDAD DE CÁDIZ Nº 10, 11519 PUERTO REAL
956 48 32 00
DIRECCION.ESI@UCA.ES
- **AUTOR:** RAÚL DÍEZ SÁNCHEZ
INGENIERO INFORMÁTICO
77171812-G
RA.DIEZSANCHEZ@ALUM.UCA.ES

Cádiz, 17 de enero de 2016

18. Presupuesto

18.1. Presupuesto para la realización del Proyecto

En la tabla ?? se incluye el presupuesto total para la realización de este Proyecto, incluyendo también los costes asociados a la implementación del sistema de demostración, IVA incluido.

Artículo	Detalles	Cantidad	Precio	Total
Raspberry Pi B	[?]	1	31,52 €/u	31,52 €
Adaptador de corriente para Raspberry Pi	[?]	1	5,78 €/u	5,78 €
Relé 40.31 12V	[?]	1	5,37 €/u	5,37 €
Relé T92S7A12-240 240V	[?]	1	17,69 €/u	17,69 €
Terminal para PCB	[?]	4	0,24 €/u	0,96 €
Tarjeta SD 8GB para Raspberry Pi	[?]	1	12,95 €/u	12,95 €
Kit de inicio Philips Hue	[?]	1	199,95 €/u	199,95 €
Latiguillo 1m RJ-45 Cat5e	[?]	1	2,06 €/u	2,06 €
Resistencia de 4,7K Ω	[?]	2	0,04 €/u	0,8 €
Arduino Mega	[?]	1	56,94 €/u	56,94 €
Arduino Ethernet Shield	[?]	1	24,01 €/u	24,01 €
Cable USB para Arduino Mega	[?]	1	3,31 €/u	3,31 €
Zumbador 230V para timbre	-	1	4,50 €/u	4,50 €
Zumbador 12V para telefonillo	[?]	1	2,71 €/u	2,71 €
Fuente de alimentación 12V	[?]	1	14,92 €/u	14,92 €
Pulsador timbre	-	1	2,75 €/u	2,75 €
Portalámparas E27	-	2	0,90 €/u	1,80 €
Total				388,02 €

Cuadro 18.1: Presupuesto para la realización del Proyecto.

18.2. Presupuesto de implantación del Proyecto

En la tabla ?? se incluye el presupuesto total para la implantación de este Proyecto, IVA incluido.

Artículo	Detalles	Cantidad	Precio	Total
Raspberry Pi B	[?]	1	31,52 €/u	31,52 €
Adaptador de corriente para Raspberry Pi	[?]	1	5,78 €/u	5,78 €
Relé 40.31 12V	[?]	1	5,37 €/u	5,37 €
Relé T92S7A12-240 240V	[?]	1	17,69 €/u	17,69 €
Terminal para PCB	[?]	4	0,24 €/u	0,96 €
Tarjeta SD 8GB para Raspberry Pi	[?]	1	12,95 €/u	12,95 €
Kit de inicio Philips Hue	[?]	1	199,95 €/u	199,95 €
Latiguillo 1m RJ-45 Cat5e	[?]	1	2,06 €/u	2,06 €
Resistencia de 4,7K Ω	[?]	2	0,04 €/u	0,8 €
Total				277,08 €

Cuadro 18.2: Presupuesto de implantación del Proyecto.

18.2.1. Mano de obra

La mano de obra asociada al Proyecto es de 35 € por instalación de dispositivo en portal, más 35 € por instalación de dispositivo en la puerta de la vivienda, más 60 € por instalación del sistema central y configuración. Esto hace un total de **130 €**.

18.2.2. Resumen del presupuesto

Los costes de presupuesto para la implantación en una vivienda real son de 277.08 €, debiéndose sumar además el coste de mano de obra, que asciende a los 130 €. Esto supone un **coste total de 407.08 €**. Se debe tener en cuenta además que este precio es el relativo a instalar 3 bombillas Hue, teniendo un coste de 52.99 € por bombilla extra a instalar [?].