enControl widget

Jordi Tejedo Jiménez
Máster Universitario en Desarrollo de Aplicaciones para Dispositivos Móviles

Francesc D'Assís Giralt Queralt
Carles Garrigues Olivella

03/01/2018

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

FICHA DEL TRABAJO FINAL

Título del trabajo:	*enControl widget*
Nombre del autor:	*Jordi Tejedo Jiménez*
Nombre del consultor/a:	*Francesc D'Assís Giralt Queralt*
Nombre del PRA:	*Carles Garrigues Olivella*
Fecha de entrega (mm/aaaa):	01/2018
Titulación::	*Máster Universitario en Desarrollo de Aplicaciones para Dispositivos Móviles*
Idioma del trabajo:	*Castellano*
Palabras clave	*Smarthome, widget, enControl*

Resumen del Trabajo

Este proyecto presenta el diseño e implementación de un *widget* para una aplicación de *smarthome* ya existente. El sistema diseñado permite llevar a cabo funciones de la aplicación de forma más rápida y directa, ahorrando de esta forma recursos. Con el trabajo realizado se pretende dotar de valor añadido a la aplicación enControl de la empresa Sensing & Control S.L. para la cual trabajo.

El sistema de *smarthome* cuenta con sensores repartidos por el hogar que envían datos a un *gateway* (Raspberry Pi) desarrollado por Sensing & Control S.L. mediante protocolo Z-Wave, Wi-Fi o Wireless M-Bus. Este *gateway* manda los datos hacia la nube, y desde allí pueden ser consultados por la aplicación o por el *widget*.

Para el desarrollo del trabajo se ha llevado a cabo una planificación, un estudio de diseño centrado en usuario, una explicación de la arquitectura global de enControl, el diseño de la arquitectura del *widget* y la implementación de este en el producto final.

La idea inicial fue desarrollar cuatro *widgets*, uno para cada apartado de enControl (confort, seguridad, control y consumos), pero la carga de trabajo que representó el primero, junto con varios escollos encontrados durante el proceso de programación, hicieron replantear la planificación.

La realización de este trabajo ha supuesto un gran aprendizaje respecto a las posibilidades del sistema operativo Android. También ha mostrado la importancia de realizar una buena planificación previa a medida que se avanza en el desarrollo.

Abstract

This project presents the design and implementation of a widget for an existing smarthome application. The designed system allows to carry out application functions faster, directly and saving resources. With this final work is intended to provide added value to the enControl application of the company Sensing & Control S.L. for which I work.

The smarthome system has sensors distributed through the home that send data to the gateway (Raspberry Pi) developed by Sensing & Control S.L. via Z-Wave, Wi-Fi or Wireless M-Bus protocol. This gateway sends the data to the cloud, and from there the data can be queried by the application or by the widget.

For the work development has been carried out a planning, a user-centered design study, an explanation of the global architecture of enControl, the design of the widget architecture and the implementation of it in the final product.

The initial idea was to develop four widgets, one for each enControl section (comfort, security, control and consumption), but the workload that represented the first and the several pitfalls encountered during the programming process made rethink the planning.

The realization of this project has meant a great learning about the Android operating system possibilities. It has also shown the importance of good advance planning as progress is made in the development.

Con este trabajo concluyo dos años y medio de master que han resultado ser muy provechosos para mi ámbito profesional. Me ha permitido trabajar para una empresa como Sensing & Control, puntera en el mundo de la tecnología y con unos compañeros fantásticos. A ellos también quiero agradecerles todos los conocimientos que me han transmitido para poder llegar a ser programador especializado en aplicaciones móviles.

Y también, porque no, me felicito a mí mismo, porque solo yo sé lo que me ha costado compaginar este máster con el trabajo, mi perro y el baloncesto entre otras cosas.

“La confianza en uno mismo es el primer secreto del éxito.”

ÍNDICE

CAPÍTULO 1: INTRODUCCIÓN 1

1.1 Contexto y justificación del Trabajo 1

1.2 Objetivos del Trabajo 4

1.3. Enfoque y método seguido 4

1.4 Planificación del trabajo 5

1.5 Breve sumario de productos obtenidos 6

1.6 Breve descripción de los otros capítulos de la memoria 6

CAPITULO 2: DISEÑO CENTRADO EN EL USUARIO 7

2.1 Usuarios y contextos de uso 7

2.2 Diseño conceptual 9

2.3 Prototipado 12

2.4 Evaluación 13

CAPÍTULO 3: ARQUITECTURA DEL SISTEMA 15

CAPÍTULO 4: DISEÑO SOFTWARE 19

4.1 Desarrollo 19
4.1.1 App Widget Provider Info 20
4.1.2 App Widget Provider 23
4.1.3 Interfaz gráfica del widget 28
4.1.4 Android Manifest 29
4.1.5 Actividad de configuración 30
4.1.6 API 33

4.2 Pruebas 35
4.2.1 Android 36
4.2.2 API 36

4.3 Futuras mejoras 39

CAPÍTULO 5: CONCLUSIONES 41

CAPÍTULO 6: GLOSARIO 42

CAPÍTULO 7: BIBLIOGRAFÍA 44

CAPÍTULO 8: ANEXOS 46

8.1 Manual de usuario 46

8.2 Código del servidor 48

LISTA DE FIGURAS

Fig. 1.1 enControl Main Activity
Fig. 1.2 enControl Comfort Activity
Fig. 1.3 enControl Security Activity
Fig. 1.4 enControl Control Activity
Fig. 1.5 enControl Consumptions Activity
Fig. 1.6 Menú accesos directos
Fig. 1.7 Diagrama de Gantt PAC 2
Fig. 1.8 Diagrama de Gantt PAC 3
Fig. 1.9 Entrega final
Fig. 2.1 Ficha Alba
Fig. 2.2 Ficha Antonio
Fig. 2.3 Diagrama UML *widgets*
Fig. 2.4 Prototipo *widget* confort
Fig. 2.5 Prototipo *widget* Seguridad
Fig. 2.6 Prototipo *widget* consumos
Fig. 2.7 Prototipo widget control
Fig. 2.8 Prototipo B *widget* consumos
Fig. 3.1 Arquitectura
Fig. 3.2 Machine Learning
Fig. 3.3 Service Bus
Fig. 3.4 *Framework* Swagger API
Fig. 3.5 *Framework* Swagger API
Fig. 4.1 App Widget Provider Info
Fig. 4.2 Tabla de medidas
Fig. 4.3 Implementación del *widget* según el prototipo
Fig. 4.4 Implementación del *widget* según la experiencia
Fig. 4.5 onUpdate()
Fig. 4.6 onEnabled()
Fig. 4.7 onDisabled()
Fig. 4.8 onDeleted()
Fig. 4.9 updateWidget()
Fig. 4.10 updateWidget()
Fig. 4.11 Interfaz gràfica
Fig. 4.12 Declaración del AppWidgetProvider
Fig. 4.13 Declaración de la actividad de configuración
Fig. 4.14 *Layout* de la actividad de configuración
Fig. 4.15 getDropDownView()
Fig. 4.16 getView()
Fig. 4.17 onClick()
Fig. 4.18 Interfaz del método en el Servidor
Fig. 4.19 Método Get InstallationSensorComfortResponse
Fig. 4.20 Interfaz del método en Android
Fig. 4.21 Método getComfortSensors()
Fig. 4.22 Clase InstallationSensorComfortResponse
Fig. 4.23 Ejemplo de uso
Fig. 4.24 Parte del código de la definición de los datos de test
Fig. 4.25 Inserción de los datos en la DB
Fig. 4.26 GetComfortSensorsForInstallation()
Fig. 4.27 Permiso
Fig. 4.28 Receptor con permiso configurado y acción SignalRBroadcastMessage permitida
Fig. 4.29 sendBroadcastMessage()
Fig. 8.1 Menú oculto
Fig. 8.2 Lista de *widgets*
Fig. 8.3 Actividad de configuración
Fig. 8.4 widget colocado
Fig. 8.5 InstallationServiceTest.cs parte 1
Fig. 8.6 InstallationServiceTest.cs parte 2
Fig. 8.7 ServiceTest.cs parte 1
Fig. 8.8 ServiceTest.cs parte 2
Fig. 8.9 ServiceTest.cs parte 3
Fig. 8.10 ServiceTest.cs parte 4
Fig. 8.11 ServiceTest.cs parte 5

CAPÍTULO 1: INTRODUCCIÓN

1.1 Contexto y justificación del Trabajo

El control y monitorización del hogar de forma remota está empezando a generar un *boom* en el mercado de las telecomunicaciones y aplicaciones móviles. Problemas como dejarse el gas o el grifo abierto, calentar la casa antes de llegar, o detectar movimiento ya tienen solución gracias a decenas de sensores que podemos encontrar en el mercado. A su vez, existen unas pocas plataformas que integran estos dispositivos para mostrarnos los datos capturados o permitirnos actuar sobre ellos a través de web o móvil.

enControl[1] [1] es una solución de *smarthome*[2] desarrollada íntegramente en Barcelona por la empresa Sensing & Control S.L. [2]. Cuenta con aplicación Web [3], Android [4] e iOS [5]. Divide su contenido en cuatro secciones bien diferenciadas:

Fig. 1.1 enControl Main Activity

- Confort → Muestra la temperatura exterior e interior del hogar, la humedad relativa y la predicción meteorológica. También incluye datos de sensores de gases como CO2 o el control del termostato y aire acondicionado de la casa.

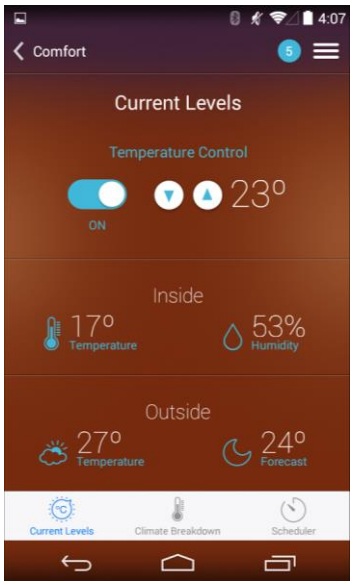

Fig. 1.2 enControl Comfort Activity

- Seguridad → Permite saber el momento en el que los inquilinos entran o salen de casa, así como ver video en directo o bloquear puertas remotamente.

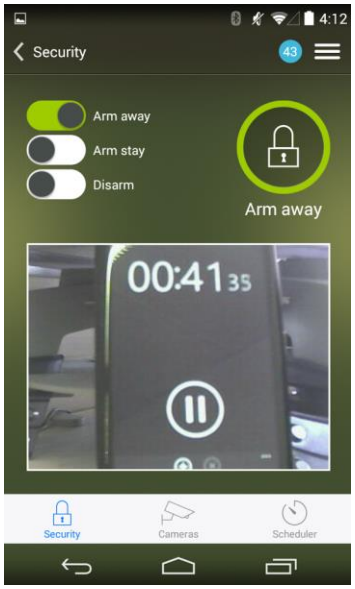

Fig. 1.3 enControl Security Activity

- Control → En esta sección se puede subir y bajar persianas, encender o apagar enchufes, o incluso encender y apagar luces.

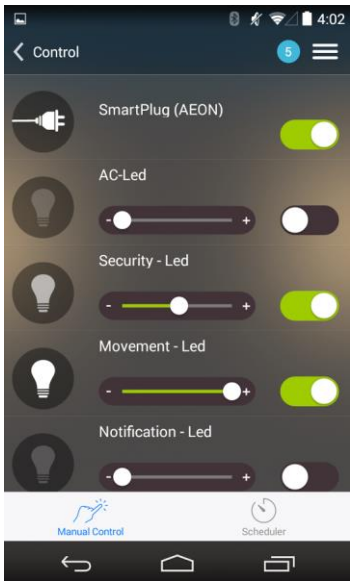

Fig. 1.4 enControl Control Activity

- Consumos → Muestra los consumos de gas, electricidad y agua del hogar, dando a la vez una predicción horaria de los mismos.

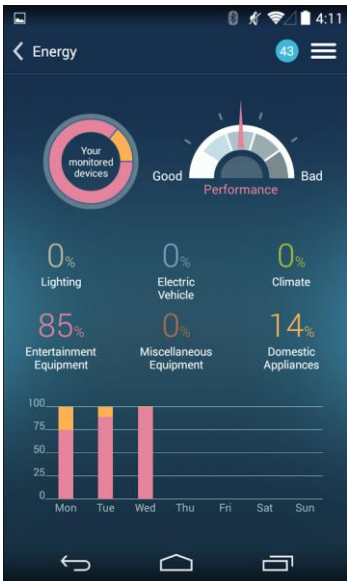

Fig. 1.5 enControl Consumptions Activity

Para tener acceso a los contenidos anteriormente mencionados el usuario tiene que hacer uso de la aplicación. Es por eso que este trabajo final de máster se centrará en desarrollar una solución para la aplicación Android de enControl, que facilite tener información detallada de lo que sucede en el hogar, y a su vez, poder interactuar con él de forma rápida y sencilla sin tener que navegar hasta las secciones deseadas.

En el siguiente link a Youtube se puede ver un video donde se explica en líneas generales el funcionamiento y el propósito de la plataforma: https://www.youtube.com/watch?v=qSYzRE9t-sw [6].

1.2 Objetivos del Trabajo

- Poder llevar a cabo las principales funciones de la *app* enControl de forma rápida y sencilla.

- Incorporar elementos innovadores para dotar de valor añadido a la aplicación.

- Reducir el uso de la *app*, tiempos de carga y llamadas a la API[3].

- Mejorar la experiencia de usuario de aquellos que para ver, por ejemplo, el consumo de agua de su hogar no quieren tener que navegar dos o tres pantallas hasta la gráfica deseada.

1.3. Enfoque y método seguido

La estrategia a seguir en este trabajo final de máster es adaptar un producto ya existente (enControl) para cumplir los objetivos marcados.

Un posible enfoque sería crear un menú de accesos directos dentro de la *app*. Nos daría acceso rápido a las funciones deseadas, pero se descartó porque no incorporaría ningún elemento innovador.

Fig. 1.6 Menú accesos directos

Otro posible enfoque seria añadir reconocimiento de voz para navegar hacia la pantalla deseada una vez reconocida la orden dada por el usuario. Sin duda es una solución que aporta valor añadido y podría parecer que mejora la experiencia de usuario, pero se consideró que el reconocimiento de voz todavía no está lo suficientemente probado y que una mala detección frustraría

demasiado al cliente. Además sería necesario acceder a la *app* para poder usar dicha función.

Finalmente, se decidió que la mejor opción sería desarrollar cuatro *widgets*[4], uno para cada sección principal de la aplicación. Pocas aplicaciones complementan sus funciones con el uso de *widgets*, por lo que parece un elemento claramente innovador. Para su uso no se debe requerir tener la aplicación abierta. Al ser accesos directos a información se disminuyen los tiempos de carga de la *app*. Por último evita tener que navegar varios niveles hasta encontrar lo que el usuario está buscando, y por lo tanto da acceso a las funciones de forma más rápida y sencilla.

1.4 Planificación del trabajo

Para realizar este trabajo se ha hecho uso de Android Studio (Java) para la programación y diseño de los *widgets*, de Visual Studio (C#) para la programación de la API y de Microsoft SQL Management Studio para el manejo de los datos.

A continuación se muestran los diagramas de Gantt con la planificación diseñada para cada PAC.

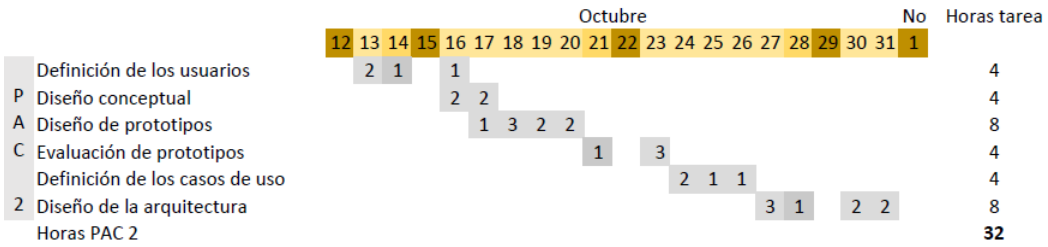

Fig. 1.7 Diagrama de Gantt PAC 2

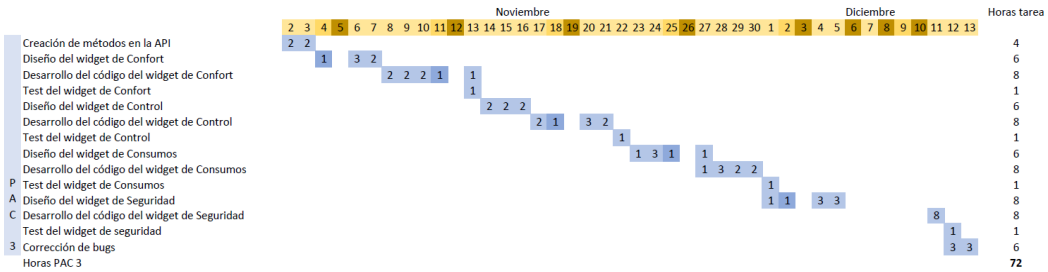

Fig. 1.8 Diagrama de Gantt PAC 3

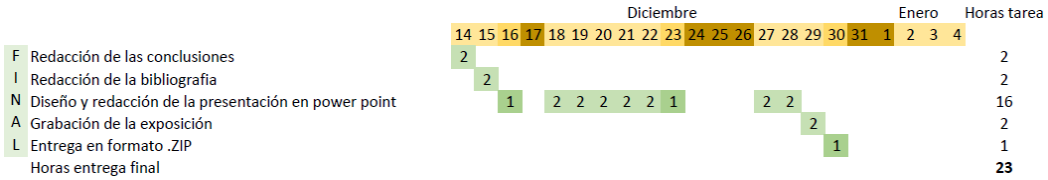

Fig. 1.9 Entrega final

Para la planificación se han tenido en cuenta días laborables, fines de semana y días festivos. En total, en la realización de PAC2, PAC3 y entrega final se emplearán un total de 127 horas, de las cuales, 11 horas serán en sábados y 116 horas en días laborables entre semana. Los festivos, domingos incluidos, no se ha previsto invertir tiempo en este trabajo salvo excepciones.

Cabe destacar una vez finalizado la programación de los *widgets* para Android habrá que hacer lo mismo para la plataforma iOS. Esa tarea se sitúa fuera del trabajo final de máster pero queda pendiente para un proyecto posterior.

1.5 Breve sumario de productos obtenidos

Al final de este trabajo se obtienen cuatro *widgets*, cada uno con sus particularidades.

El *widget* de confort mostrará el último dato capturado de los sensores de confort que el usuario escoja. Por ejemplo, la temperatura y humedad interior y la temperatura y humedad exterior.

El *widget* de seguridad mostrará el estado de los sensores de seguridad que el usuario escoja. Por ejemplo, detección o no detección de presencia en el comedor o detección o no detección de humo en la cocina. También se podrán activar o desactivar las alarmas de la casa.

El *widget* de control mostrará acciones disponibles sobre los sensores actuables que el usuario escoja. Por ejemplo, subir o bajar persiana o encender o apagar luz.

Por último, el *widget* de consumos mostrará datos en tiempo real de los consumos del hogar. Por ejemplo, una gráfica de los consumos del día de hoy.

1.6 Breve descripción de los otros capítulos de la memoria

Los capítulos que se muestran a continuación en la memoria del trabajo son:

- Diseño centrado en el usuario → Se hace un estudio detallado de los potenciales usuarios de la aplicación, se diseñan y se evalúan los prototipos y se definen los casos de uso.

- Arquitectura del sistema → Se muestra un esquema acompañado de una explicación de la arquitectura que hay detrás de la aplicación enControl junto con documentación de la API. También se explica cuál es el lugar y función de los *widgets* en un sistema tan complejo.

- Diseño del software → Se muestran detalles técnicos referentes al software que se ha desarrollado para este trabajo, como por ejemplo, funciones claves, buenas prácticas y test.

CAPITULO 2: DISEÑO CENTRADO EN EL USUARIO

2.1 Usuarios y contextos de uso

Los usuarios focales de este producto son propietarios de inmuebles, tanto casas como pisos, apartamentos o locales entre otros. En función de lo que determine el propietario, los usufructuarios pueden ser usuarios focales si tienen pleno derecho al uso de la aplicación, usuarios secundarios si solo tienen acceso a ella en casos determinados o usuarios no prioritarios si no están autorizados a usarla. También se puede dar el caso de que los propietarios sean simples promotores, como por ejemplo en hoteles.

En general los usuarios nombrados no tienen por qué tener conocimientos tecnológicos muy elevados, pero si un interés al respecto ya que es precisamente eso lo que les habrá llevado a usar la aplicación. Se trata de personas comprendidas en un rango de edad grande y de ámbitos muy diversos, desde padres o madres de familia, hasta hijos, dueños de negocios o clientes de un hotel. De esta manera, habrá usuarios de perfiles muy diferentes, pero todos compartirán la misma inquietud, por lo que el diseño tendrá que ser orientado a ejecutar acciones sobre el inmueble de la forma más sencilla posible teniendo en cuenta las complejidades de cada función.

A continuación se muestran dos fichas de usuarios focales con sus correspondientes descripciones de escenario:

Nombre: Alba
Edad: 25
Profesión: Enfermera

Descripción de la persona:
Alba trabaja en la unidad de traumatología del Hospital de Bellvitge en turnos de 12 horas, de 8:00h a 20:00h. Vive con su pareja y sus dos perros en Esplugas de Llobregat y se desplaza en coche todos los días para ir a trabajar. Hace uso habitual del móvil para acceder a redes sociales, escuchar música o pedir comida para llevar. Pese a que tiene ordenador, son pocas veces a la semana las que lo usa. Vive en una casita pequeña con jardín. En invierno es una zona fría y húmeda y cuando llega a casa tarda hasta una hora en conseguir calentar el hogar, lo que le provoca más de un constipado.

Descripción del escenario:
Es un lunes a las 6:50 y Alba se levanta para ir a trabajar. Se prepara y a las 7:20 sale de casa, no sin antes haber encendido el deshumidificador al mínimo para minimizar la humedad en casa hasta que vuelva. Durante su jornada laboral duda de si se ha dejado la nevera abierta debido a las prisas del desayuno, algo que ya le mantiene preocupada todo el día. A las 20:20 llega a casa y comprueba que la nevera estaba cerrada. Hace frio así que decide poner la calefacción. Aprovechando que tardará un rato en calentar el inmueble sale a pasear a los perros. A la vuelta por fin puede sentarse tranquila y disfrutar del confort del hogar.

Fig. 2.1 Ficha Alba

Nombre: Antonio
Edad: 57
Profesión: Mecánico

Descripción de la persona:
Antonio trabaja de encargado de mantenimiento en Aguas de Barcelona de 8:00h a 18:00h con 2 horas para comer. Vive con su mujer y su hijo menor en Esplugas de Llobregat y se desplaza caminando todos los días para ir a trabajar e ir a comer a casa. Hace un uso básico del móvil, aunque cada vez está más familiarizado. La *tablet* la usa a menudo para navegar por internet. El ordenador lo usa exclusivamente para descargar contenido por internet. Vive en un piso muy soleado, y en verano se queja del calor que hace en casa por la tarde hasta que él llega de trabajar y coloca el toldo.

Descripción del escenario:
Es viernes a las 6:00 y Antonio se levanta para ir a trabajar. El lunes próximo es festivo así que al salir del trabajo pasará a buscar a su mujer y su hijo y se marcharán de fin de semana. Él siempre cuando va a estar unos días fuera cierra ventanas y puertas para evitar la entrada de ladrones, pero no tiene ningún método que le asegure que no entre nadie en casa. No es hasta el lunes a última hora del día cuando regresa y puede estar tranquilo de que todo sigue igual.

Fig. 2.2 Ficha Antonio

2.2 Diseño conceptual

A continuación se valoran los pros y contras del uso de enControl por parte de los dos protagonistas de las fichas de usuario del apartado anterior en base a sus descripciones de escenario.

Si Alba usara enControl podría incorporar un sensor de humedad y un sensor de enchufe a su habitación. El sensor de humedad detectaría cuando esta se encuentra por debajo del límite. En el caso de que eso se produjera enControl mandaría una orden al sensor de enchufe para encender el deshumidificador. Existen otras posibles soluciones, por ejemplo, si Alba no quiere que el deshumidificador se encienda cuando ella está en casa solo tendría que informar a la plataforma de la localización de su casa. Su móvil se encargaría de avisar a la plataforma de cuando Alba entra en el perímetro de la casa y de cuando sale. Cuando se encuentre fuera y la humedad sea inferior a lo programado se activará el sensor de enchufe. El punto más problemático seria la inversión inicial en dos o tres sensores que le podrían costar alrededor de los 100€.

Antonio para la seguridad de su casa podría incorporar una cámara junto con un sensor de movimiento. El sensor de movimiento detectaría si alguien entra en casa y la cámara se encargaría de capturar video en directo y fotos en el

caso de que hubiera movimiento en el hogar. Si eso sucediera llegaría una alerta al móvil de Antonio. Si se quisiera hacer la solución más robusta se podría incorporar una sirena a casa para que no solo llegara la alarma al móvil del cliente si no que esta sonara para disuadir al ladrón. Al igual que en el caso de Alba, el punto más problemático seria la inversión inicial, pero se considera que tratándose de seguridad del hogar sería una inversión asumible.

Como se puede ver, enControl puede llevar a cabo más de 30 funciones distintas ya que son muchos los sensores integrados dentro la plataforma. Existen desde los que nos permiten mover persianas venecianas hasta los que nos detectan la presencia de CO en casa, pasando por otros más sencillos, como por ejemplo, de detección de puerta abierta o cerrada o los que se usarían para los casos de Alba y Antonio. Es por ello que en este apartado solo se van a listar aquellas funciones que tengan que ver con el desarrollo de los *widgets* que se lleva a cabo en este trabajo.

- Visualización de datos capturados → Según el sensor podremos ver datos de humedad, temperatura, luminosidad, presión, consumos…

- Cambiar estado de seguridad del hogar → Es lo más parecido a la función de activar o desactivar la alarma convencional de casa cuando el inquilino llega. En este caso la función ofrece la posibilidad de armar la casa (nivel de seguridad máximo), armarla de forma parcial (solo algunas alarmas se activarían) o desarmarla (estado pensado para cuando el inquilino está en casa).

- Ver última foto capturada por la cámara principal de la instalación → No solo se puede ver la última foto capturada si no que clicando en ella podríamos acceder al historial de fotos realizadas y al video en directo.

- Actuación sobre sensores de control → Esto incluye cualquier orden que se pueda llevar a cabo sobre sensores que lo permitan. Por ejemplo, un sensor de CO2 no permite recibir ninguna orden porque tan solo es un medidor, pero un sensor LED permite recibir órdenes de apagado y encendido y de intensidad de la luz.

A continuación se muestra el diagrama UML de los *widgets* con actores, flujos y procesos:

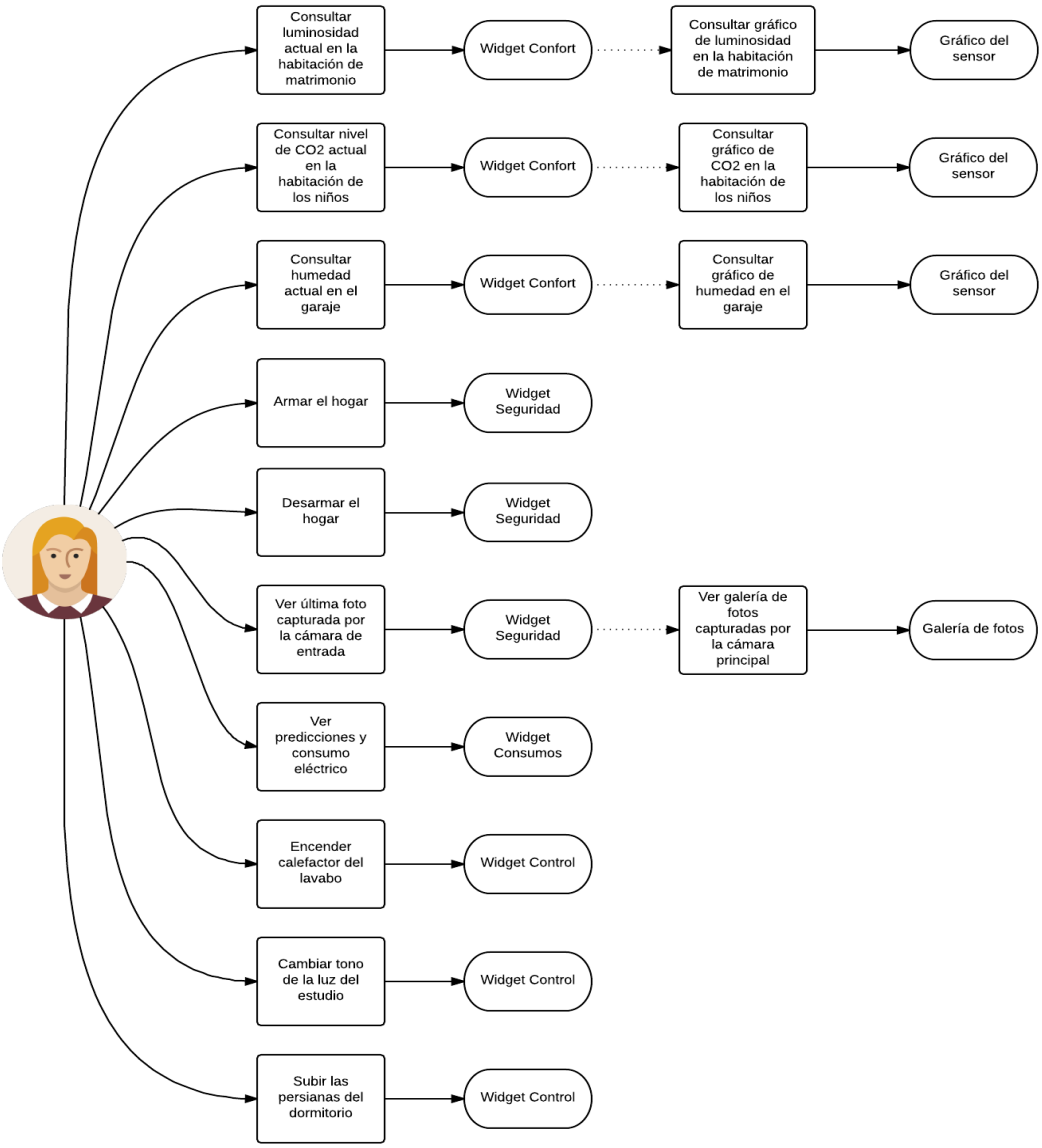

Fig. 2.3 Diagrama UML *widgets*

Las cajas cuadradas representan algunos ejemplos de procesos que puede llevar a cabo el usuario. Las cajas redondeadas son donde terminan los procesos. En la mayoría de casos serán *widgets*. Las flechas indican los flujos, y las líneas discontinuas significan un salto de los *widgets* a la *app*. Si el usuario lleva a cabo algún proceso precedido por una línea discontinua quiere decir que será redirigido a la *app* para continuar con la funcionalidad que está solicitando.

Cabe destacar que en este trabajo solo se realiza programación de los *widgets*, los cuales se basan en una *app* ya existente. Existe la posibilidad de que la *app* cambie durante el proceso de desarrollo de este proyecto y algunas funcionalidades precedidas por líneas discontinuas dejen de funcionar o no tengan el comportamiento esperado.

2.3 Prototipado

Para la realización de los prototipos se ha seguido la línea de estilos de las *apps* y de la web de enControl. A continuación se pueden ver los diseños plasmados sobre la pantalla principal de un dispositivo Android.

Fig. 2.4 Prototipo *widget* confort

Fig. 2.5 Prototipo *widget* seguridad

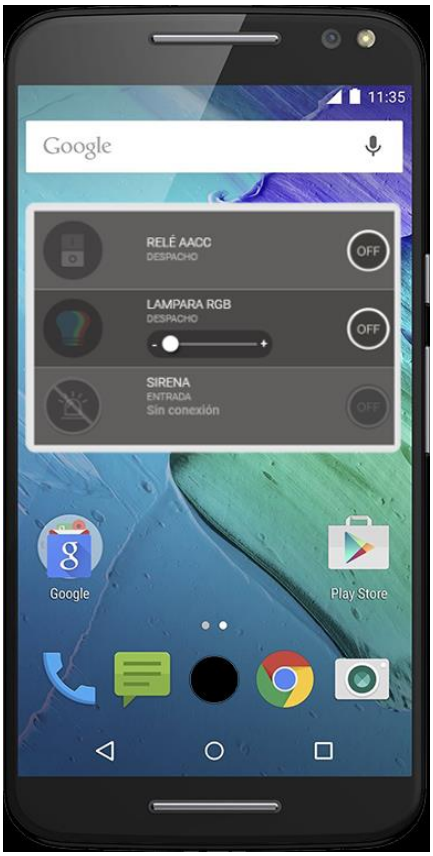

Fig. 2.6 Prototipo *widget* consumos **Fig. 2.7** Prototipo *widget* control

2.4 Evaluación

La evaluación de los prototipos a largo plazo se basará en los resultados de la experiencia de usuario. A priori se puede decir que los diseños del *widget* de confort y del de control no deberían presentar problemas ya que son intuitivos, claros y prácticos. Se considera que sus tamaños de letra son adecuados. En el caso del de control, su altura variará en función de los sensores que el usuario configure para manejar desde el *widget*.

En cuanto al *widget* de seguridad, lo único que plantea dudas es el tamaño de la imagen capturada. No puede ser mucho más grande ya que rompería la armonía de la pantalla. De todos modos, la intención no es que el usuario vea la imagen claramente desde el *widget*, si no que sepa que se ha capturado una foto nueva y clique en ella para verla a través de la *app* en un tamaño superior.

Por último, el *widget* de consumos ha planteado varias dudas en su diseño. Se considera que a diferencia del de confort, el de consumos debe mostrar una gráfica más detallada para que el usuario tenga a mano los datos históricos y las predicciones, que son puntos de valor añadido para la aplicación. En el caso de que la experiencia de usuario muestre que una gráfica de tamaño comprimido no es útil se podría optar por el diseño de la figura 2.6.

Fig. 2.8 Prototipo B *widget* consumos

En este diseño en vez de una gráfica se mostrarían datos de los tres posibles consumos del hogar (gas, electricidad y agua). Los círculos serán verdes en caso de tener un consumo similar a la predicción, azules en caso de tener un consumo inferior a lo previsto y rojos en caso de estar consumiendo por encima de lo normal para ese hogar. Es una forma de ayudar al usuario a tomar conciencia de su consumo de energía y poder así ahorrar en la factura a final de mes.

CAPÍTULO 3: ARQUITECTURA DEL SISTEMA

La plataforma consiste en una serie de servicios en la nube a los cuales se comunican los entornos web y móvil (Android e iOS) mediante una API, ya sea para obtener datos o realizar acciones. El propio servidor interactúa periódicamente o bajo demanda con el *gateway*[5] que el usuario tiene físicamente en su dominio (casa, negocio…) el cual hace de puente con los sensores y cámaras.

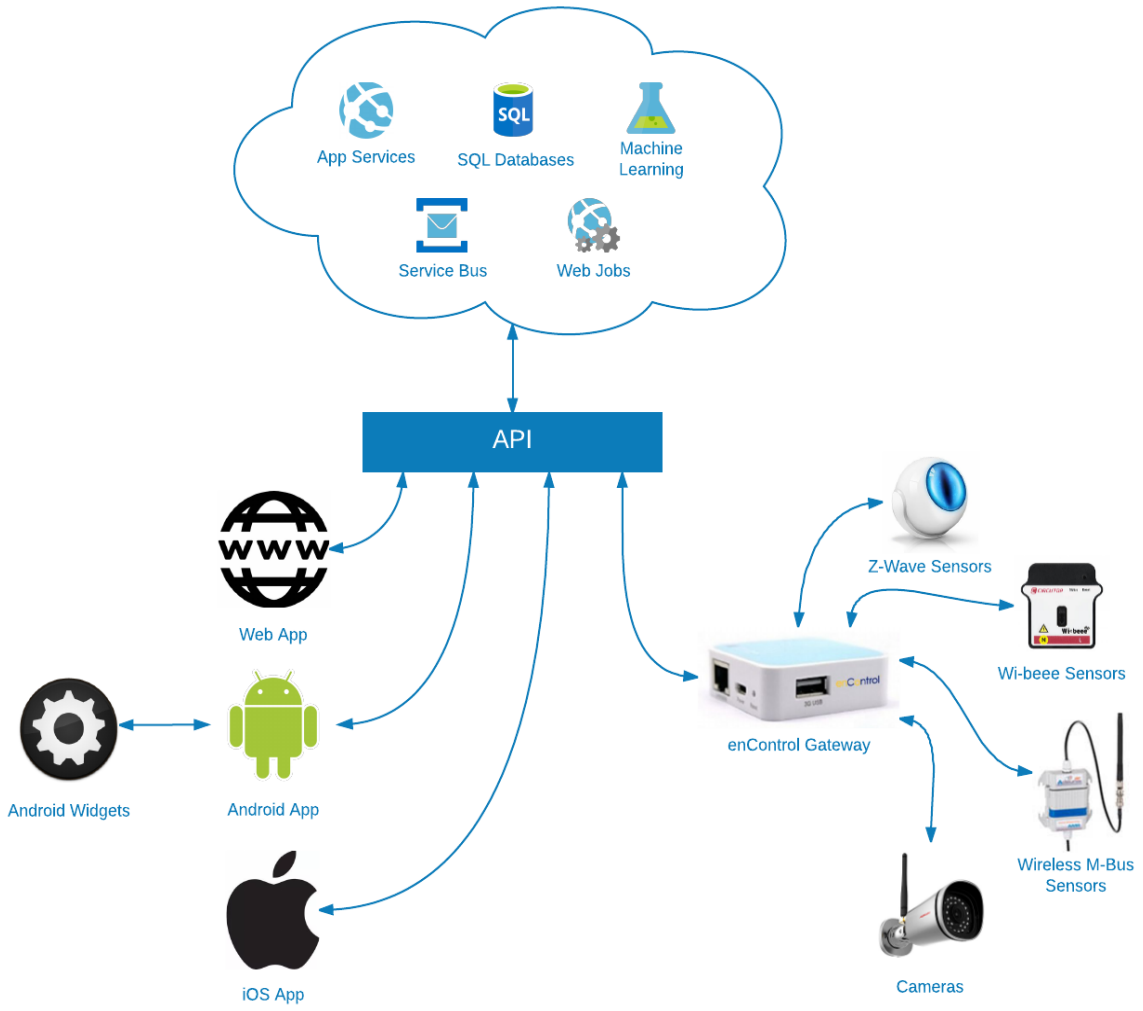

Fig. 3.1 Arquitectura

El *backend*[6] del sistema se encuentra alojado en la nube mediante los servidores de Azure [7]. La plataforma Azure proporciona una serie de servicios que se utilizan en enControl. Algunos de los más importantes son:

- SQL Databases → Las bases de datos SQL que se manejan en la aplicación también están alojadas de forma remota.

- Machine Learning [8] → Es la herramienta utilizada para calcular las predicciones de consumos. Requiere de un mínimo conocimiento de inteligencia artificial pero facilita mucho los cálculos. Consiste en un programa mediante el cual a partir de un esquema con cajas puedes crear tu experimento predictivo. En la siguiente imagen se puede ver un ejemplo.

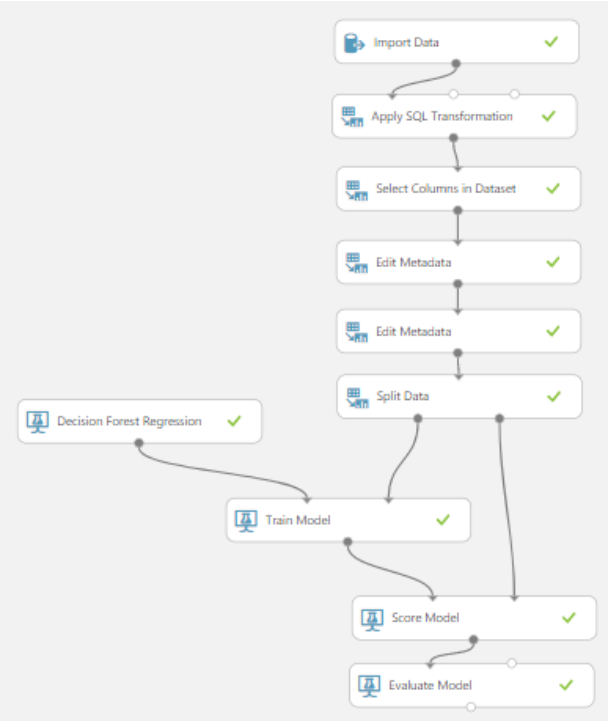

Fig. 3.2 Machine Learning

- Service Bus → Mediante esta herramienta la plataforma crea una cola para cada instalación. A esas colas, entre otras muchas cosas, se envían mensajes con los valores capturados por cada sensor. Esos mensajes son consumidos por las aplicaciones web, Android e iOS para mostrar los datos. Una vez los mensajes se consumen son eliminados.

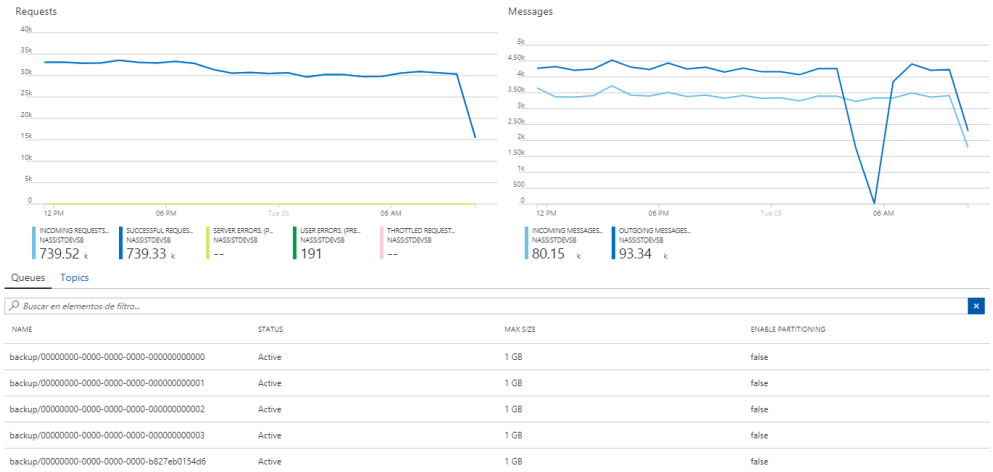

Fig. 3.3 Service Bus

- Web Jobs → Consisten en pequeños programas desarrollados en C# que se pueden ejecutar de forma continua o bajo demanda. enControl los usa para sacar estadísticas de consumos o para detectar *gaps*[7] u *outliers*[8] en los valores de los sensores entre otras cosas.

La API es la manera de interactuar y llevar a cabo funciones de la plataforma. A parte de usarla para las aplicaciones web, Android e iOS, también puede ser usada por clientes que lo soliciten para hacer uso de los servicios de enControl desde su propia aplicación. Como todo el servidor, está programada en C#, pero para hacer su uso más amigable se ha incorporado un *framework*[9] llamado Swagger [9] que permite documentarla, testearla y usarla por terceros. El link a la API de enControl es el siguiente: https://encontrol.io/api/swagger-ui/ [10].

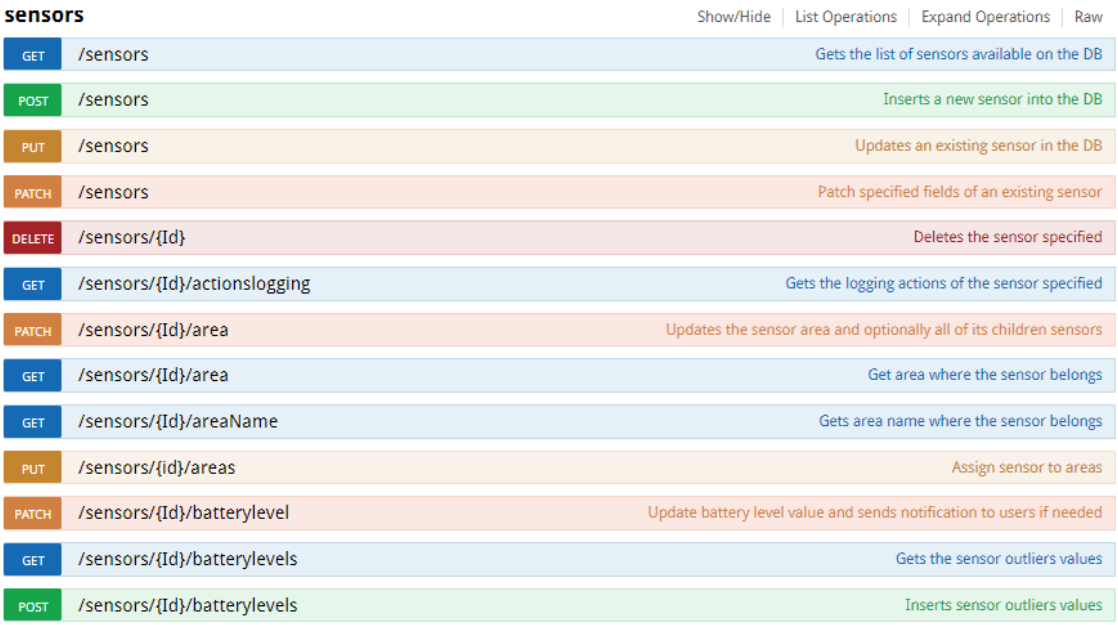

Fig. 3.4 *Framework* Swagger API

Fig. 3.5 *Framework* Swagger API

En Sensing & Control S.L. también hemos desarrollado un código de ejemplo tanto en C# como en Java para que resulte sencillo para otros programadores familiarizarse con la API. Los ejemplos se pueden encontrar en Github en el siguiente link: https://github.com/Sensing-Control-DevTeam [11]

El *gateway* de enControl consiste en una Raspberry Pi 3 [12] con un módulo Razberry [13] incorporado para poder emitir y recibir mensajes con el protocolo[10] Z-Wave [14]. Está programado íntegramente por Sensing & Control S.L. Su función es recibir órdenes del servidor y ejecutarlas sobre los dispositivos o recibir datos de los dispositivos y enviarlos al servidor.

Las aplicaciones Android e iOS se han desarrollado de forma nativa debido a la complejidad del sistema y las funciones que se requieren llevar a cabo desde ellas. Junto con la web son las interfaces propuestas por Sensing & Control S.L. para que el usuario final interactúe con el sistema. Todas las peticiones al servidor se hacen a través de la API.

Por último, los *widgets*, al depender de la aplicación Android funcionan de igual manera que ella. Sin embargo, la idea es que cuando ésta es cerrada por el usuario puedan funcionar de forma autónoma.

CAPÍTULO 4: DISEÑO SOFTWARE

A la hora de poner en práctica los conceptos planteados en apartados anteriores se han encontrado algunos problemas que han impedido cumplir con la planificación de la PAC3.

Para el desarrollo del código de cada *widget* se estipularon un total de ocho horas y finalmente se necesitaron alrededor de cuarenta. Probablemente, con un poco de experiencia previa se hubiera empleado menos tiempo, pero la curva de aprendizaje ha resultado más grande de lo previsto.

La corrección de *bugs* también estaba planeado que no llevara más de seis horas. La previsión quedó muy lejos de las veinte horas reales que se empleó para este punto. No es que los errores cometidos fueran muy graves, si no que la búsqueda de soluciones fue costosa y requería conocimientos complementarios.

Por último, la planificación más errónea fue la de los test. Una sola hora era completamente irreal si se querían llevar a cabo test con cara y ojos. Además, si estos test revelaban errores que provocaban que la *app* se tuviera que volver a probar una vez solucionados el tiempo se multiplicaba. Se calcula que fueron ocho horas de test si consideramos también la programación correspondiente de estos.

A continuación se muestra el proceso de desarrollo y las decisiones que se han ido tomando para solventar dichos inconvenientes.

4.1 Desarrollo

Para crear un *widget* se necesita:

1. Un archivo XML que defina sus propiedades. A este se le denomina comúnmente App Widget Provider Info [15] y define el tamaño del *widget* en pantalla, su frecuencia de actualización y su *layout*[11] entre otras cosas.

2. Una clase[12] que defina los métodos básicos que permitan interactuar mediante programación con él en función de los eventos capturados. Esta tendrá que ser una extensión de AppWidgetProvider [16] si queremos realizar acciones cuando se actualice, habilite, deshabilite y elimine el *widget*. La extensión AppWidgetProvider es a su vez una extensión de BroadcastReceiver [17]. En el caso de querer personalizar los eventos capturados la clase podrá extender directamente de BroadcastReceiver.

3. Un archivo XML que defina su vista inicial.

4. Declararlo en el archivo AndroidManifest.xml de la aplicación.

Adicionalmente se puede implementar una actividad[13] de configuración. Esta es una actividad opcional que se inicia cuando el usuario agrega el *widget* y le permite modificar la configuración del mismo en el momento de la creación.

De aquí en adelante se usará el *widget* de confort como ejemplo. El resto no se han podido desarrollar debido a una planificación muy optimista que se ha unido a los problemas anteriormente mencionados y que se irán describiendo.

4.1.1 App Widget Provider Info

A continuación se muestran el archivo XML de propiedades.

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="80dp"
    android:minHeight="80dp"
    android:updatePeriodMillis="0"
    android:initialLayout="@layout/widget_comfort"
    android:configure="com.sensingcontrol.android.widgets.ComfortWidgetConfigurationActivity"
    android:resizeMode="none"
    android:widgetCategory="home_screen"
    android:previewImage="@drawable/comfort_widget_preview">
</appwidget-provider>
```

Fig. 4.1 App Widget Provider Info

Los valores de minWidth y minHeight especifican el espacio mínimo que el *widget* consume por defecto. La pantalla de inicio de los dispositivos Android coloca los *widgets* en función de una cuadricula de celdas, que tienen un alto y un ancho definidos. Si los valores para el ancho o alto mínimo no coinciden con las dimensiones de las celdas se redondean al tamaño de celda más cercano.

Muchos teléfonos cuentan con una cuadricula de 4x4 mientras que las *tablets* pueden llegar a ofrecer cuadriculas de 8x7. Es por eso que se desaconseja diseñar *widgets* que ocupen más que 4x4 ya que dejará de ser útil en algunos modelos. Las medidas de las celdas, así como los márgenes entre cada una varían entre dispositivos. Se puede usar la tabla de la figura 4.2 para estimar aproximadamente las dimensiones mínimas.

# of Cells (Columns or Rows)	Available Size (dp) (minWidth or minHeight)
1	40dp
2	110dp
3	180dp
4	250dp
...	...
n	70 × *n* − 30

Fig. 4.2 Tabla de medidas

Teniendo en cuenta las directrices de diseño de Android [18] y el prototipo inicial primero se pensó en un *widget* que ocupara 4x1.

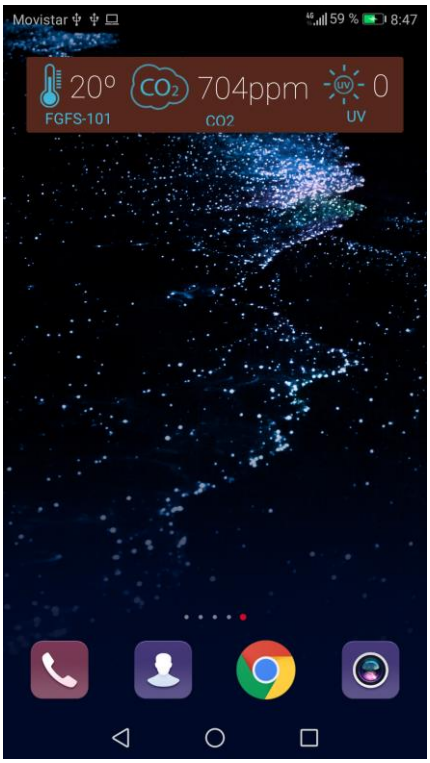

Fig. 4.3 Implementación del *widget* según el prototipo

Como se puede observar, y pese a lo previsto en la evaluación de los prototipos, la implementación no es la deseada. El ancho es demasiado pequeño para medidas de sensores que ocupen más de lo normal y el alto es muy justo para mostrar el nombre del sensor sin que quede cortado. Además, si el usuario tiene o quiere visualizar menos de tres sensores quedaría un espacio vacío. En el caso de que quisiera ver más de tres sensores no le quedará más remedio que añadir otro *widget* de 4x1 repitiendo el mismo problema.

Para solucionarlo finalmente se ha optado por una medida de 2x2, que según la tabla de la figura 4.2 equivale a 110dp de ancho y 110dp de alto. De ahí el valor que aparece en las propiedades minWidth y minHeight.

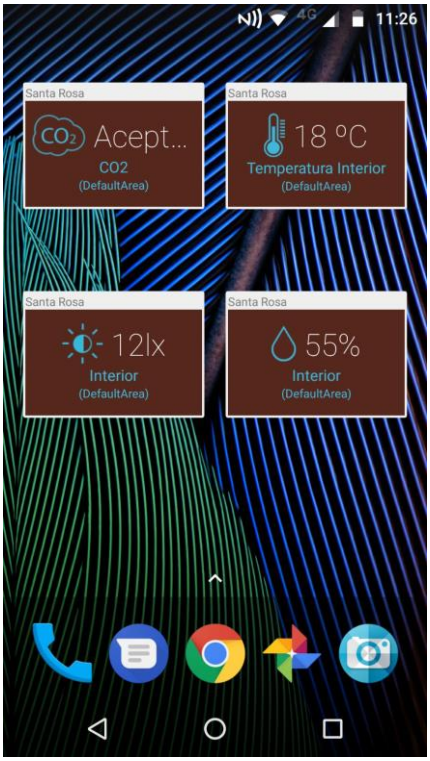

Fig. 4.4 Implementación del *widget* según la experiencia

En este caso, en cada *widget* se muestra solo un sensor, permitiendo así que el usuario pueda añadir tantos como quiera sin tener que depender de que la cantidad sea múltiplo de tres. El tamaño se ha ampliado para permitir valores más largos, así como para mostrar el nombre del sensor acompañado del área entre paréntesis (lugar de la instalación donde está ubicado: dormitorio, cocina, recepción…) y el nombre de la instalación, por si el usuario tiene más de una instalación asignada y quiere mezclar sensores de ambas.

La propiedad updatePeriodMillis del AppWidgetProviderInfo define con qué frecuencia se debe solicitar una actualización. Android sugiere que se actualice con la menor frecuencia posible para conservar la batería del dispositivo. En el caso de este *widget* la propiedad se ha definido a 0, ya que es la manera de hacer que nunca se actualice de forma periódica. Algunos sensores envían datos cada dos minutos, otros una vez al día, por lo que no tiene sentido forzar actualizaciones periódicamente porque habrá sensores que no habrán renovado su último valor.

Tener siempre disponible medidas en tiempo real de cada sensor es una prioridad en este trabajo. La mejor forma de lograrlo es mediante el mismo servicio que usa la aplicación enControl para actualizar sus valores. Está desarrollado íntegramente por Sensing & Control S.L. y se basa en la librería SignalR [19] de ASP.NET. Esta biblioteca facilita el proceso de añadir funcionalidades en tiempo real a aplicaciones web o móviles, es decir, permite al servidor enviar información al cliente sin tener que esperar a que el cliente realice una petición. El servidor tiene un servicio desarrollado que se encarga de enviar datos a diferentes *hubs*. El cliente solo tiene que subscribirse a esos *hubs* para recibirlos.

En este punto se encuentra uno de los problemas encontrados a lo largo de la fase de desarrollo. El servicio[14] existente en Android permite recibir datos si nos subscribimos a alguno de los diferentes *hubs*. Sin embargo, cuando la aplicación se cierra el servicio se detiene y por consiguiente los *widgets* dejan de recibir medidas en tiempo real. Este está configurado para que cuando algún evento lo cierre se vuelva a poner en marcha automáticamente, pero a la práctica esto no está funcionando. El problema se encontró en la fase final de desarrollo del *widget* de confort y ha sido uno de los quebraderos de cabeza. La solución propuesta se encuentra explicada en el apartado de futuras mejoras de la memoria, pero ha resultado imposible desarrollarla por la cantidad de trabajo que representa en la fase final del proyecto.

El valor de la propiedad initalLayout especifica la ruta donde encontrar el XML que usaremos para la vista del *widget*. En este caso se encuentra en la carpeta layout con nombre widget_comfort.xml.

En este *widget* ha hecho falta implementar una actividad de configuración que se encuentra definida en la propiedad configure con nombre ComfortWidgetConfigurationActivity.java. Los detalles de esta se explican más adelante el apartado 4.1.5.

La propiedad resizeMode especifica en que eje se puede cambiar el tamaño del *widget*. Se puede configurar para que sean redimensionables de forma horizontal, vertical, ambos o ninguna. Para este caso se ha pensado en dejar el tamaño del *widget* de forma fija ya que no aportaría ningún valor añadido teniendo en cuenta lo que queremos mostrar. En el caso de querer dar la opción al usuario de hacerlo más grande se puede añadir código para a la vez que el tamaño aumenta ir añadiendo nuevos elementos visuales.

La propiedad widgetCategory define en que pantalla será visible el *widget*. Para versiones de Android 5.0 y superiores existe la única opción de mostrarlo en la pantalla de inicio (home_screen). Para versiones anteriores se puede mostrar también en la pantalla de bloqueo (keyguard) o ambas. Dado el propósito de este *widget* será solo visible en la pantalla de inicio.

Por último, la propiedad previewImage indica que imagen se mostrará en el selector de *widgets* de Android a modo de previsualización.

Existen más propiedades opcionales que no ha sido necesario definirlas porque no procedía.

4.1.2 App Widget Provider

A esta clase se le ha llamado ComfortWidget y extiende de AppWidgetProvider. No se ha necesitado personalizar ningún evento capturado así que se ha descartado usar la extensión de BroadcastReceiver.

Primeramente se ha sobrescrito la función onUpdate(). La documentación dice que se invoca cuando el usuario agrega el *widget* a la aplicación, salvo si se ha declarado una actividad de configuración. En ese caso será la actividad de configuración la encargada de hacer la primera actualización del *widget*, y a partir de entonces el método se ejecutará según el intervalo definido en la propiedad updatePeriodMillis del App Widget Provider Info. En la práctica se ha comprobado que esto no es así y es posible que se trate de un error en la documentación de Android. La experiencia muestra que la función onUpdate() se ejecuta siempre antes que la actividad de configuración. Esto consume recursos innecesariamente, ya que no hay nada que actualizar porque el *widget* no se ha definido y puede generar *bugs* en el código. En el portal de StackOverflow se pueden encontrar varios hilos al respecto del problema, este es uno de ellos: https://stackoverflow.com/questions/9144262/android-widget-show-configuration-activity-before-widget-is-added-to-the-screen [20]. También se copia a continuación el enlace de la incidencia presentada en Google: https://issuetracker.google.com/issues/36924475#c0 [21].

Para solucionar el problema definido en el párrafo anterior se ha agregado un booleano a SharedPreferences que dirá si el *widget* que se está tratando ha pasado por la actividad de configuración o no. En caso negativo se saltará esa actualización.

```
@Override
public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
    final int N = appWidgetIds.length;

    SharedPreferences prefs = context.getSharedPreferences("ComfortWidgetPrefs",
Context.MODE_PRIVATE);

    // Perform this loop procedure for each App Widget that belongs to this provider
    for (int i=0; i<N; i++) {
        int appWidgetId = appWidgetIds[i];

        // Before update, check if appWidgetId exist
        if(prefs.contains(appWidgetId + "_id")) {
            updateWidget(context, appWidgetManager, appWidgetId);
        }
    }
}
```

Fig. 4.5 onUpdate()

El método onEnabled() se llama cuando sea crea el App Widget Provider por primera vez. Es decir, si el usuario crea dos *widgets* de confort solo se ejecutará durante la creación del primero. Es el sitio idóneo para registrarse en el servicio a partir del cual se recibirán los valores de los sensores. Se llama NewSignalRJavaClient y lo que hace es suscribirse a los *hubs*[15] de todas las instalaciones que el usuario tenga asignado. Por consiguiente también se suscribe a los *hubs* de todos los sensores que tenga asignados una instalación. Los datos que recibe los envía a sus *endpoints* mediante mensajes *broadcast* locales. No es la forma más eficiente de hacerlo, pero como el desarrollo de dicho servicio no forma parte de este proyecto no se ha cambiado. Sin embargo en el apartado de futuras mejoras se propone otra forma de hacerlo sin tener que estar suscrito a *hubs* que no interesan.

```
@Override
public void onEnabled(Context context){
   LocalBroadcastManager.getInstance(context).registerReceiver(mSignalRReceiver, new
IntentFilter("SignalRBroadcastMessage"));
}
```

Fig. 4.6 onEnabled()

De forma similar, el método onDisabled() se invoca cuando se elimina el último *widget* de confort. Aquí es donde nos desregistramos del servicio y eliminamos los datos del tipo SharedPreferences guardados durante las creaciones de los *widgets*.

```
@Override
public void onDisabled(Context context){
   LocalBroadcastManager.getInstance(context).unregisterReceiver(mSignalRReceiver);

   SharedPreferences prefs = context.getSharedPreferences("ComfortWidgetPrefs",
Context.MODE_PRIVATE);
   SharedPreferences.Editor editor = prefs.edit();
   editor.clear().commit();
}
```

Fig. 4.7 onDisabled()

También se ha implementado el método onDeleted(). Se llama cuando se han eliminado uno o más *widgets* de confort. En este caso dentro de la función se ha añadido el código necesario para eliminar del SharedPreferences los valores correspondientes al sensor borrado. El resto de datos seguirán existiendo para poder actualizar los *widgets* que queden activos.

```
@Override
public void onDeleted(Context context, int[] appWidgetIds) {
   // Search all appWidgetId info saved in SharedPreferences and remove
   for (int i=0; i < appWidgetIds.length; i++) {
      SharedPreferences prefs = context.getSharedPreferences("ComfortWidgetPrefs",
Context.MODE_PRIVATE);
      String sensorId = prefs.getString(appWidgetIds[i] + "_id", "");

      SharedPreferences.Editor editor = prefs.edit();
      editor.remove(sensorId + "_mAppWidgetId");
      editor.remove(appWidgetIds[i] + "_installationName");
      editor.remove(appWidgetIds[i] + "_id");
      editor.remove(appWidgetIds[i] + "_name");
      editor.remove(appWidgetIds[i] + "_area");
      editor.remove(appWidgetIds[i] + "_type");
      editor.remove(appWidgetIds[i] + "_lastValue");
      editor.remove(appWidgetIds[i] + "_selected");
      editor.remove(appWidgetIds[i] + "_baseIconName");
      editor.remove(appWidgetIds[i] + "_units");
      editor.commit();
   }
}
```

Fig. 4.8 onDeleted()

Hay otra función más dentro de la clase ComfortWidget que no extiende de AppWidgetProvider pero lo complementa y es indispensable para su correcto funcionamiento. Se llama updateWidget() y se encarga de actualizar los datos que el *widget* muestra por pantalla. Para ello crea una variable del tipo RemoteViews [22]. Su función es modificar una vista que forme parte de otro proceso. Esta nos permite manejar la mayoría de elementos de una vista remota, aunque no todos, como por ejemplo los Spinner. De ahí que los *widgets* no admitan todo tipo de diseños.

```
public static void updateWidget(Context context, AppWidgetManager appWidgetManager, int appWidgetId) {
   SharedPreferences prefs = context.getSharedPreferences("ComfortWidgetPrefs",
Context.MODE_PRIVATE);
   String installationName = prefs.getString(appWidgetId + "_installationName",  "");
   String id = prefs.getString(appWidgetId + "_id",  "");
   String name = prefs.getString(appWidgetId + "_name", "");
   String area = prefs.getString(appWidgetId + "_area", "");
   String type = prefs.getString(appWidgetId + "_type", "");
   Double lastValue = Double.parseDouble(prefs.getString(appWidgetId + "_lastValue", "0"));
   Boolean selected = prefs.getBoolean(appWidgetId + "_selected", false);
   String baseIconName = prefs.getString(appWidgetId + "_baseIconName", "");
   String units = prefs.getString(appWidgetId + "_units", "");

   ComfortWidgetConfigurationActivity.ViewModel sensor = new
ComfortWidgetConfigurationActivity.ViewModel(id, name, area, type, lastValue, selected, baseIconName,
units);

   Boolean isCelsius = PreferencesUtil.getBooleanValue(context.getApplicationContext(), "isCelsius", true);

   RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.widget_comfort);

   views.setTextViewText(R.id.installation_name, prefs.getString(appWidgetId + "_installationName",  ""));
   views.setTextViewText(R.id.sensor_name, prefs.getString(appWidgetId + "_name", ""));
   views.setTextViewText(R.id.sensor_area, "("+prefs.getString(appWidgetId + "_area", "")+")");

   if (prefs.getString(appWidgetId + "_units", "").equals("º"))
   {
      if (isCelsius)
      {
         views.setTextViewText(R.id.sensor_measurement,
TemperatureConverter.RenderTemperatureValue(sensor.getLastValue(), isCelsius) + " " + sensor.getUnits() +
"C");
      }
      else {
         views.setTextViewText(R.id.sensor_measurement,
TemperatureConverter.RenderTemperatureValue(sensor.getLastValue(), isCelsius) + " " +sensor.getUnits() +
"F");
      }
   }
   else if (sensor.getType().equals("CO2")) {
      views.setTextViewText(R.id.sensor_measurement, ValuesRenderer.RenderCO2AQIValue(context,
sensor.getLastValue()));
   }
   else if (sensor.getType().equals("Ultraviolet")) {
      views.setTextViewText(R.id.sensor_measurement, ValuesRenderer.RenderUltravioletValue(context,
sensor.getLastValue()));
   }
   else {
      views.setTextViewText(R.id.sensor_measurement, sensor.getLastValue().intValue() + sensor.getUnits());
   }

   views.setImageViewResource(R.id.sensor_icon, context.getResources().getIdentifier("drawable/" +
sensor.getBaseIconName() + "_ico" , null, MainActivity.PACKAGE_NAME));

   appWidgetManager.updateAppWidget(appWidgetId, views);
}
```

Fig. 4.9 updateWidget()

Por último, se ha tenido que crear una variable del tipo BroadcastReceiver para capturar los mensajes enviados por el servicio NewSignalRJavaClient. La variable incluye una función onReceive() que se ha sobrescrito para *parsear* los datos del sensor recibido y modificarlos en el SharedPreferences que

posteriormente será usado en la función updateWidget() para hacer la actualización.

```
private BroadcastReceiver mSignalRReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        String messageSubscription = intent.getStringExtra("MessageSubscription");

        if (messageSubscription.contentEquals("receiveNewSensorData")) {
            String sensorId = intent.getStringExtra("SensorId");
            String value = intent.getStringExtra("MessageValue");

            SharedPreferences prefs = context.getSharedPreferences("ComfortWidgetPrefs",
Context.MODE_PRIVATE);
            if (prefs.contains(sensorId + "_mAppWidgetId")) {
                int mAppWidgetId = prefs.getInt(sensorId + "_mAppWidgetId", 0);
                SharedPreferences.Editor editor = prefs.edit();
                editor.putString(mAppWidgetId + "_lastValue", value);
                editor.apply();

                AppWidgetManager widgetManager = AppWidgetManager.getInstance(context);

                updateWidget(context, widgetManager, mAppWidgetId);
            }
        }
    }
};
```

Fig. 4.10 updateWidget()

4.1.3 Interfaz gráfica del widget

Su diseño es bastante sencillo. Cuenta con un ImageView para el icono del tipo de sensor y cuatro TextView, uno para el valor de la medida, otro para el nombre del sensor, otro para el área a la cual pertenece el sensor y el último para el nombre de la instalación a la cual pertenece el sensor. También incorpora diferentes *layouts* para alinear los elementos. No se muestra el código XML debido a su escaso interés en la memoria pero se puede encontrar en el .ZIP del proyecto.

Por lo general los *widgets* no deben extenderse a los bordes de la pantalla y se recomienda agregar márgenes alrededor de su marco. Sin embargo yo no los he añadido porque a partir de Android 4.0 reciben automáticamente relleno entre el marco del *widget* y el cuadro delimitador del mismo. Por otro lado, se ha añadido un borde gris de 2dp para estilizarlo.

Fig. 4.11 Interfaz gràfica

4.1.4 Android Manifest

En el manifiesto de la aplicación se han declarado dos elementos nuevos, uno correspondiente al App Widget Provider y otro referente a la actividad de configuración.

```
<receiver android:name=".widgets.ComfortWidget">
   <intent-filter>
      <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
   </intent-filter>

   <meta-data
      android:name="android.appwidget.provider"
      android:resource="@xml/widget_comfort_info" />
</receiver>
```

Fig. 4.12 Declaración del AppWidgetProvider

El <intent-filter> incluye un elemento <action> que especifica que el App Widget Provider acepta la transmisión de acciones APPWIDGET_UPDATE. Esta es la única acción que debe declararse explícitamente, todas las demás transmisiones se envían automáticamente. Además se ha declarado una variable <meta-data> donde se especifica cual es el App Widget Provider Info.

```
<activity
   android:name=".widgets.ComfortWidgetConfigurationActivity"
   android:configChanges="orientation"
   android:label="Comfort Widget"
   android:screenOrientation="portrait">
   <intent-filter>
      <action android:name="android.appwidget.action.APPWIDGET_CONFIGURE" />
   </intent-filter>
</activity>
```

Fig. 4.13 Declaración de la actividad de configuración

La actividad de configuración se ha declarado como una actividad normal en el AndroidManifest.xml. Sin embargo, será lanzada por el *widget* con la acción APPWIDGET_CONFIGURE, por lo que la actividad debe aceptarla.

4.1.5 Actividad de configuración

Como se ha comentado en otros apartados, esta será la actividad encargada de configurar el *widget* de confort. En ella aparece un desplegable donde se puede seleccionar cualquiera de las instalaciones que estén asignadas al usuario con el que se ha iniciado sesión en la aplicación. Por consiguiente, según la instalación seleccionada aparecerán los sensores de confort disponibles para monitorizar.

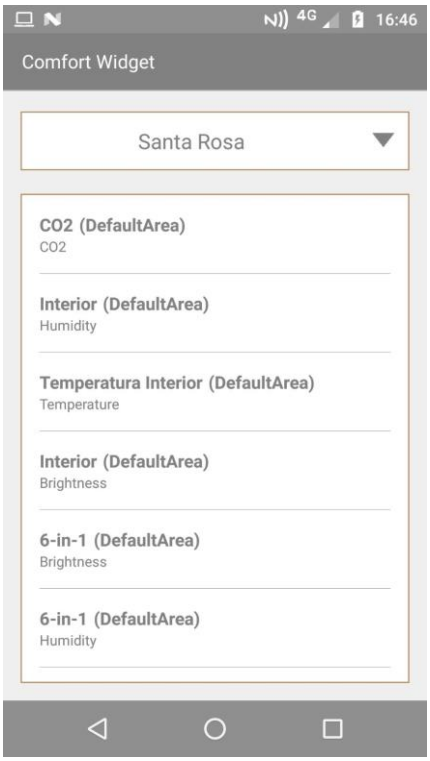

Fig. 4.14 *Layout* de la actividad de configuración

Para la lista de instalaciones se ha creado un Spinner con un adaptador personalizado. El Spinner tendrá dos tipos de vista, la normal y la desplegada. Para personalizar ambas se han sobrescrito los métodos getDropDownView() y getView() del SpinnerAdapter.

```
@Override
public View getDropDownView(int position, View convertView, ViewGroup parent) {
    TextView txt = new TextView(activity.getBaseContext());
    txt.setPadding(10, 10, 10, 10);
    txt.setTextSize(16);
    txt.setGravity(Gravity.CENTER);
    txt.setText(installationsList.get(position).getName());
    txt.setTextColor(Color.parseColor("#818181"));
    txt.setBackgroundResource(R.drawable.installations_dropdown_spinner_background);
    return  txt;
}
```

Fig. 4.15 getDropDownView()

```
@Override
public View getView(int i, View view, ViewGroup viewgroup) {
    TextView txt = new TextView(activity.getBaseContext());
    txt.setGravity(Gravity.CENTER);
    txt.setPadding(10, 10, 10, 10);
    txt.setTextSize(18);
    txt.setCompoundDrawablesWithIntrinsicBounds(0, 0, R.drawable.group_arrow_down_grey, 0);
    txt.setText(installationsList.get(i).getName());
    txt.setTextColor(Color.parseColor("#818181"));
    return  txt;
}
```

Fig. 4.16 getView()

Para la lista de sensores se ha utilizado una ListView con un adaptador personalizado. Se ha decidido que la actualización del *widget* se lance cuando el usuario seleccione con el dedo uno de los sensores. En la mayoría de casos esta acción se llevaría a cabo en la actividad de configuración, pero debido a la interacción del usuario con la lista se ha incorporado dentro del adaptador del ListView. A continuación se muestra la función onClick() del OnClickListener() de los elementos de la lista.

```
@Override
    public void onClick(View v) {
        Boolean isCelsius = PreferencesUtil.getBooleanValue(activity.getApplicationContext(), "isCelsius",
true);
        ViewHolder holder = (ViewHolder) v.getTag();
        ComfortWidgetConfigurationActivity.ViewModel sensor = sensorsList.get(holder.position);

        RemoteViews views = new RemoteViews(activity.getPackageName(), R.layout.widget_comfort);
        views.setTextViewText(R.id.installation_name, installationName);
        views.setTextViewText(R.id.sensor_name, sensor.getName());
        views.setTextViewText(R.id.sensor_area, "("+sensor.getAreaName()+")");

        if (sensor.getUnits().equals("°"))
        {
            if (isCelsius)
            {
                views.setTextViewText(R.id.sensor_measurement,
TemperatureConverter.RenderTemperatureValue(sensor.getLastValue(), isCelsius) + " " + sensor.getUnits() +
"C");
            }
            else {
                views.setTextViewText(R.id.sensor_measurement,
TemperatureConverter.RenderTemperatureValue(sensor.getLastValue(), isCelsius) + " " +sensor.getUnits() +
"F");
            }
        }
        else if (sensor.getType().equals("CO2")) {
            views.setTextViewText(R.id.sensor_measurement, ValuesRenderer.RenderCO2AQIValue(activity,
sensor.getLastValue()));
        }
        else if (sensor.getType().equals("Ultraviolet")) {
            views.setTextViewText(R.id.sensor_measurement, ValuesRenderer.RenderUltravioletValue(activity,
sensor.getLastValue()));
        }
        else {
            views.setTextViewText(R.id.sensor_measurement, sensor.getLastValue().intValue() +
sensor.getUnits());
        }

        views.setImageViewResource(R.id.sensor_icon, activity.getResources().getIdentifier("drawable/" +
sensor.getBaseIconName() + "_ico" , null, MainActivity.PACKAGE_NAME));

        SharedPreferences prefs = activity.getSharedPreferences("ComfortWidgetPrefs",
Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = prefs.edit();
        editor.putInt(sensor.getId() + "_mAppWidgetId", mAppWidgetId);
        editor.putString(mAppWidgetId + "_installationName", installationName);
        editor.putString(mAppWidgetId + "_id", sensor.getId());
        editor.putString(mAppWidgetId + "_name", sensor.getName());
        editor.putString(mAppWidgetId + "_area", sensor.getAreaName());
        editor.putString(mAppWidgetId + "_type", sensor.getType());
        editor.putString(mAppWidgetId + "_lastValue", sensor.getLastValue().toString());
        editor.putBoolean(mAppWidgetId + "_selected", sensor.isSelected());
        editor.putString(mAppWidgetId + "_baseIconName", sensor.getBaseIconName());
        editor.putString(mAppWidgetId + "_units", sensor.getUnits());
        editor.commit();

        AppWidgetManager.getInstance(activity).updateAppWidget(mAppWidgetId, views);

        Intent result = new Intent();
        result.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, mAppWidgetId);
        activity.setResult(RESULT_OK, result);
        activity.finish();
    }
```

Fig. 4.17 onClick()

En las últimas cinco líneas de código se puede ver el proceso que se sigue para actualizar o crear el *widget* y dar por finalizada la actividad de configuración. Primero se obtiene una instancia del AppWidgetManager [23] y acto seguido se ejecuta la actualización a partir del identificador del *widget* y el RemoteView rellenado. Por último, se crea un Intent donde se añadirá el identificador del *widget* actualizado o creado junto con el código RESULT_OK, que indica que está listo para ser mostrado en la página de inicio del dispositivo, y se cierra la actividad.

Es importante decir que una buena práctica ha sido añadir un setResult(RESULT_CANCELED) en las primeras líneas de la función onCreate() de la actividad de configuración. Esto asegura que si el usuario cierra la pantalla sin haber seleccionado ningún sensor el *widget* se dará por descartado. Esto último junto con el resto de código de la actividad y los adaptadores se puede consultar en el .ZIP del proyecto.

4.1.6 API

Para la API de enControl se ha tenido que crear un método que nos devolviese una lista con exclusivamente los sensores de confort.

```
[Route("/installations/{Id}/sensors/comfort", Summary = "Gets a list of
comfort sensors linked to the specified installation", Verbs = "GET")]
    public class InstallationSensorsComfort :
IReturn<InstallationSensorsComfortResponse>
    {
        [ApiMember(Name = "Id", Description = "Installation id", ParameterType
= "path", DataType = "Guid", IsRequired = true)]
        public Guid Id { get; set; }
    }
```

Fig. 4.18 Interfaz del método en el Servidor

Como se puede observar en la imagen el método será del tipo GET y se podrá acceder a él a través de una ruta, que al fin y al cabo será lo que se use en la aplicación para hacer peticiones. La función hará una búsqueda en la base de datos y devolverá exclusivamente los sensores del tipo luminosidad, CO2, humedad, termostato, termostato principal, temperatura, transmisor ACIR, relé y ultravioleta.

```
public async Task<InstallationSensorsComfortResponse>
Get(InstallationSensorsComfort request)
        {
            await RequestVerificationHub.VerifyRequest(request, this);

            return new InstallationSensorsComfortResponse
            {
                Sensors = await RetryAsync.Do(() =>
Db.SelectAsync<SensorWithAreaName>(Db.From<Installation>()
                        .Join<Floorplan>((i, f) => f.InstallationId ==
request.Id)
                        .Join<Floorplan, Area>((f, a) => f.Id ==
a.FloorplanId)
                        .Join<Area, AreaNode>((a, an) => a.Id == an.AreaId)
                        .Join<AreaNode, Node>((an, n) => an.NodeId == n.Id)
                        .Join<Node, NodeSensor>((n, ns) => n.Id == ns.NodeId)
                        .Join<NodeSensor, Sensor>((ns, s) => ns.SensorId ==
s.Id)
                        .Where<Sensor>(s => Sql.In(s.Type,
SensorType.Brightness, SensorType.CO2, SensorType.Humidity,
SensorType.Thermostat, SensorType.MainThermostat, SensorType.Temperature,
SensorType.ACIRTransmitter, SensorType.ThermostatRelay,
SensorType.Ultraviolet))
                        .Select("DISTINCT Sensor.*, Area.Name AreaName")))
            };
        }
```

Fig. 4.19 Método Get InstallationSensorComfortResponse

Para facilitar la comunicación con la API del servidor se ha creado también una interfaz en Android.

```
@Route(Path="/installations/{Id}/sensors/comfort", Verbs="GET")
public static class InstallationSensorsComfort implements IReturn<InstallationSensorsComfortResponse>
{
   @ApiMember(Name="Id", Description="Installation id", ParameterType="path", DataType="Guid",
IsRequired=true)
   public UUID Id = null;

   public UUID getId() { return Id; }
   public InstallationSensorsComfort setId(UUID value) { this.Id = value; return this; }
   private static Object responseType = InstallationSensorsComfortResponse.class;
   public Object getResponseType() { return responseType; }
}
```

Fig. 4.20 Interfaz del método en Android

El método, a diferencia del de la API del servidor, no hará una llamada a la base de datos, si no que recibirá la respuesta y la almacenará en una variable del tipo InstallationSensorComfortResponse.

```
public void getComfortSensors(String installationId, int retries,
SStackCustomCallback<InstallationSensorsComfortResponse> callback) {
    InstallationSensorsComfort request = new InstallationSensorsComfort();
    request.setId(UUID.fromString(installationId));

    executeRequest(request, HttpMethods.Get, retries, callback);
}
```

Fig. 4.21 Método getComfortSensors()

InstallationSensorComfortResponse es una clase que contiene un ArrayList<SensorWithAreaName> y los métodos correspondientes get() y set().

```
public static class InstallationSensorsComfortResponse extends ResponseBase
{
    public ArrayList<SensorWithAreaName> Sensors = null;

    public ArrayList<SensorWithAreaName> getSensors() { return Sensors; }
    public InstallationSensorsComfortResponse setSensors(ArrayList<SensorWithAreaName> value) {
this.Sensors = value; return this; }
}
```

Fig. 4.22 Clase InstallationSensorComfortResponse

Por último, se puede ver un ejemplo de uso de la interfaz Android para comunicarse con la API del servidor.

```
SStackWebApiConsumer.getInstance().getComfortSensors(installationId, 3, new
SStackCustomCallback<dto.InstallationSensorsComfortResponse>() {
    @Override
    public void onSuccess(dto.InstallationSensorsComfortResponse responseObject) {
        sensorsList.clear();

        for(dto.SensorWithAreaName sensor : responseObject.getSensors()) {
            sensorsList.add(new ViewModel(sensor.getId(), sensor.getName(), sensor.getAreaName(),
sensor.getType(), sensor.getValue(), false, sensor.getBaseIconName(), sensor.getUnitsForAverageGraph()));
        }

        adapter = new ComfortWidgetSensorsListViewAdapter(sensorsList,
ComfortWidgetConfigurationActivity.this, mAppWidgetId, installationName);
        sensorsListView.setAdapter(adapter);
    }

    @Override
    public void onFailure(Exception e) {

    }
});
```

Fig. 4.23 Ejemplo de uso

4.2 Pruebas

En este apartado se muestran las pruebas llevadas a cabo tanto en los *widgets* como en el del servidor.

4.2.1 Android

En los *widgets* se han realizado pruebas de usabilidad. Los encargados de ello han sido Alberto Fernandez y Borja Raurich, dos compañeros de Sensing & Control S.L. que suelen estar al cargo de los test de enControl día a día. Yo también he validado dichas pruebas, pero ni mucho menos tiene el mismo valor porque el desarrollador de un producto a la hora de testearlo suele estar condicionado por el profundo conocimiento que tiene a cerca de él. Es fundamental que los test sean llevados a cabo también por terceras personas.

Se han usado tres teléfonos, un Nexus 4 con Android 5.0, un Moto 5 Plus y un Huawei P10, ambos con Android 7. El Nexus 4 ha mostrado un comportamiento algo perezoso con los *widgets*, aunque es algo que ya se preveía ya que le sucede lo mismo con la aplicación debido a que se trata un teléfono antiguo al borde de la obsolescencia. Los teléfonos Nexus 5 plus y Huawei P10 han funcionado perfectamente.

Tanto Alberto como Borja no han cometido ningún error en el procedimiento a la hora de generar un *widget*. Ha resultado intuitivo y amigable. El tiempo requerido para concluir la actividad ha sido inferior a veinte segundos y se han sentido satisfechos con el resultado. Durante el proceso de testeo han surgido propuestas de mejoras que se muestran en el apartado correspondiente.

En todo momento se han tenido en cuenta los problemas presentados durante el apartado 4.1. Sería conveniente realizar unos test nuevos una vez solventados.

4.2.2 API

Para las llamadas a la API de enControl se han realizado test en NUnit [24]. Es un *framework* para todos los lenguajes .Net. Se creó a partir de JUnit [25] para Java.

Lo primero que se ha hecho es definir un usuario junto con una instalación ficticia con tres dispositivos asignados. Las instalaciones se pueden dividir en plantas, las plantas contienen áreas, las áreas nodos, y los nodos sensores. Esto es algo heredado de la arquitectura de enControl. No merece la pena entrar en más detalles, pero es importante tenerlo en cuenta porque al crear la instalación también se han tenido que crear sus plantas, áreas y nodos correspondientes.

```
// Instantiating some test data
            testUser = new UserAuth { Id = 1, UserName = "admin", Email =
"admin@test.admin", CreatedDate = DateTime.Now, ModifiedDate = DateTime.Now };
            testInstallation = new Installation { Id = Guid.Parse("00000000-
0000-0000-0000-000000000001"), Type = "SensingControl", CreationDate =
DateTime.Now, LastModificationDate = DateTime.Now, ActivationDate =
DateTime.Now };
            testUserInstallation = new UserInstallation { Id = 1,
InstallationId = Guid.Parse("00000000-0000-0000-0000-000000000001"), UserId =
1, UserInGeofence = false };
            testFloorplan = new Floorplan { Id = 1, InstallationId =
Guid.Parse("00000000-0000-0000-0000-000000000001"), CreationDate =
DateTime.Now, LastModificationDate = DateTime.Now };
            testArea = new Area { Id = 1, Name = "DefaultArea", Description =
"Default Area", FloorplanId = 1, CreationDate = DateTime.Now,
LastModificationDate = DateTime.Now };
            testNodeCam = new Node { Id = 1, Name = "Test Node Cam",
Description = "Test Node Cam", CreationDate = DateTime.Now,
LastModificationDate = DateTime.Now };
            testNodeSens = new Node { Id = 2, Name = "Test Node Sens",
Description = "Test Node Sens", CreationDate = DateTime.Now,
LastModificationDate = DateTime.Now };
            testAreaNodeCam = new AreaNode { AreaId = 1, NodeId = 1 };
            testAreaNodeSens = new AreaNode { AreaId = 1, NodeId = 2 };
            testCameraNode = new CameraNode { CameraId = "00000000-0000-0000-
0000-000000000001", NodeId = 1 };
            testNodeSensor = new NodeSensor { SensorId = "00000000-0000-0000-
0000-000000000001", NodeId = 2 };
            testNodeSensor = new NodeSensor { SensorId = "00000000-0000-0000-
0000-000000000002", NodeId = 2 };

            testSensor = new Sensor
            {
                Id = "00000000-0000-0000-0000-000000000001",
                Type = SensorType.EnergyMeter.ToString(),
                Name = "Test",
                Value = 5,
                Status = "ok",
                LastDateValue = DateTime.UtcNow,
                LastDateStatus = DateTime.UtcNow,
                Configuration = "",
                IsVirtual = false,
                DeviceStatus = DeviceStatus.ok.ToString(),
                DeviceStatusDate = DateTime.UtcNow
            };
…
…
…
```

Fig. 4.24 Parte del código de la definición de los datos de test

Seguidamente, se guardan en diferentes tablas dentro de la base de datos de test sobre la cual se realizarán las consultas.

```
// Create required tables for the tests and fill them with test data (if
needed)
// This is recreated for every test using an in-memory sqlite database (easier
and faster than mocking db connection)
using (var db = appHost.Resolve<IDbConnectionFactory>().Open())
{
// We don't strictly follow SStack DB primary key design conventions so we
must create DB tables via script rather than OrmLite methods

       db.ExecuteSql(ddlScript);

       db.Insert(testUser);
       db.Insert(testInstallation);
       db.Insert(testUserInstallation);
       db.Insert(testFloorplan);
       db.Insert(testArea);
       db.Insert(testNodeSens);
       db.Insert(testNodeCam);
       db.Insert(testAreaNodeCam);
       db.Insert(testAreaNodeSens);
       db.Insert(testCamera);
       db.Insert(testSensor);
       db.Insert(testSensor2);
       db.Insert(testCameraNode);
       db.Insert(testNodeSensor);
}
```

Fig. 4.25 Inserción de los datos en la DB

Una vez definidos los datos sobre los que se realizarán pruebas se ha desarrollado el método que testeará la llamada a la API que nos devuelve los sensores de confort asignados a una instalación.

```
[Test]
        public async Task GetComfortSensorsForInstallation()
        {
            List<SensorWithAreaName> comfortSensors = (await
installationService.Get(new InstallationSensorsComfort { Id =
testInstallation.Id })).Sensors;

            Assert.AreEqual(1, comfortSensors.Count());
            Assert.AreEqual(testSensor2, comfortSensors.Where(x => x.Id ==
testSensor2.Id.ToString()));
        }
```

Fig. 4.26 GetComfortSensorsForInstallation()

Primero obtiene los sensores de confort correspondientes a la instalación de test. Seguidamente se comprueba que la respuesta solo contenga un sensor, ya que de los tres dispositivos enlazados a la instalación solo uno corresponde al apartado de confort. Por último se verifica que este sea igual al que esperábamos obtener. Si cualquiera de estas dos comprobaciones fallara el test informaría de un error.

4.3 Futuras mejoras

En el apartado 4.1.1 se expone el problema que supone tener que trabajar para recibir datos en tiempo real de los sensores con un servicio [26] que se detiene cuando la aplicación se cierra. Más concretamente lo que sucede es que el servicio está vinculado con la actividad que lo crea, y por lo tanto, cuando esta actividad se destruye, sucede lo mismo con el servicio. Este se reinicia automáticamente, ya que así ha sido configurado, pero por un motivo que desconozco los *hubs* de SignalR no vuelven a funcionar si la aplicación está cerrada. El uso de dicha librería dentro de este servicio no ayuda a encontrar una solución fácil, ya que al ser adaptadas de .NET no disponen de documentación expresa para Java. Creo que con más tiempo para testear el problema se podría llegar a una solución que no implicara tener que cambiar de librería, ya que con la aplicación abierta está funcionando correctamente. En caso contrario se podría valorar la opción de cambiar SignalR por Atmosphere [27].

En el apartado 4.1.2 se explica el comportamiento anómalo de la función onUpdate(). Para evitar su ejecución innecesaria se implementó un booleano. Otra posible solución sería extender la clase ComfortWidget.java de BroadcastReceiver en vez de AppWidgetProvider. Esto permitiría usar únicamente una función onReceive() para todos los eventos capturados. Dentro de esta se trataría cada uno de forma personalizada. También se podría dejar de usar mensajes *broadcast* locales que requieren de un registro previo para poder recibirlos, para pasar a usar mensajes *broadcast* sin registro. Estos son menos seguros, ya que pueden ser capturados por cualquier receptor. Sin embargo, esto se podría solventar creando un permiso especial para los mensajes que deseemos.

```
<uses-permission android:name="com.sensingcontrol.android.permission.WIDGET" />
```

Fig. 4.27 Permiso

```
<receiver android:name=".widgets.ComfortWidget"
   android:permission="com.sensingcontrol.android.permission.WIDGET">
   <intent-filter>
      <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />

      <action android:name="SignalRBroadcastMessage" />
   </intent-filter>

   <meta-data
      android:name="android.appwidget.provider"
      android:resource="@xml/widget_comfort_info" />
</receiver>
```

Fig. 4.28 Receptor con permiso configurado y acción SignalRBroadcastMessage permitida

El envío de mensajes *broadcast* quedaría como en la figura 4.29.

```
private void sendBroadcastMessage(String messageSubscription, String value, String sensorId) {
    if(!isStopping) {
        Intent intent = new Intent("SignalRBroadcastMessage");
        intent.putExtra("MessageSubscription", messageSubscription);
        intent.putExtra("MessageValue", value);

        if (!sensorId.contentEquals("")) {
            intent.putExtra("SensorId", sensorId);
        }

        //Code for local broadcast --> LocalBroadcastManager.getInstance(context).sendBroadcast(intent);
        sendBroadcast(intent, Manifest.permission.WIDGET);
    }
}
```

Fig. 4.29 sendBroadcastMessage()

El servicio con SignalR que se usa se podría mejorar evitando suscribirse a todos los *hubs* de una instalación (instalación, sensores asignados, cámaras asignadas…) de golpe para suscribirse solo a aquellos que fueran necesarios. Actualmente se tienen que filtrar muchos mensajes que se recogen de los *hubs* y no tienen interés para los *widgets*.

Por último, Durante las pruebas de usabilidad se planteó la posibilidad de que no solo se recibiera por SignalR el nuevo valor del sensor, sino también el área o el nombre del mismo si cambiaran. Los test de uso quedarían perfectamente complementados con pruebas unitarias JUnit.

CAPÍTULO 5: CONCLUSIONES

Este proyecto consigue plasmar el proceso de desarrollo de un *widget* desde su diseño y planificación hasta su desarrollo y testeo. Además, pese a no poder solucionar algunos de los problemas de adaptación con el código de enControl ya existente, se proponen soluciones a todos ellos.

Durante su desarrollo se ha aprendido el funcionamiento de los *widgets* dentro del sistema operativo Android y su ciclo de vida. También se ha profundizado en conceptos sobre servicios, procesos y subprocesos, así como test de usabilidad y NUnit.

Se considera que se ha conseguido cumplir con todos los objetivos, sin olvidar que finalmente solo se ha podido desarrollar un solo *widget*. Al inicio se siguió la planificación realizada paso a paso, pero al poco tiempo se pudo ver que la correspondiente a la PAC3 no era adecuada para la carga de trabajo que suponía. Sin duda, durante ese tramo, se han dedicado al proyecto más horas de las planificadas dando como resultado menos código desarrollado. Mucho de ese tiempo se ha empleado en buscar soluciones y replantear ideas. El principal error ha sido querer abarcar cuatro desarrollos siendo esta mi primera experiencia con *widgets*. Lo correcto hubiera sido asignar más horas a los primeros desarrollos e ir disminuyéndolas en los siguientes a medida que se consolidaban nuevos conocimientos. También hay que añadir que unas horas estipuladas en la planificación para leer y estudiar la forma de integrar este proyecto dentro del código ya existente de enControl hubiera prevenido futuros problemas. Por otro lado, Todo esto ha provocado que hubiera que introducir cambios para garantizar el éxito del trabajo.

Ha quedado pendiente el desarrollo de los *widgets* de seguridad, control y consumos. Previamente habría que dedicar tiempo a implementar un nuevo servicio que nos proporcionara los datos de los sensores en vivo independientemente de si la aplicación está abierta o cerrada. Solo en el caso de que el usuario no estuviera identificado no sería necesario tener el servicio corriendo.

CAPÍTULO 6: GLOSARIO

1. enControl → Nombre dado por Sensing & Control S.L. al sistema de *smarthome* desarrollado por ellos mismos.

2. Smarthome → También conocido como domótica. Se denomina así a los sistemas capaces de automatizar una vivienda o edificación de cualquier tipo, aportando servicios de gestión energética, seguridad, bienestar y comunicación, y que pueden estar integrados por medio de redes interiores y exteriores de comunicación, cableadas o inalámbricas, y cuyo control goza de cierta ubicuidad, desde dentro y fuera del hogar.

3. API (Application Programming Interface) → Es un conjunto de subrutinas, funciones y métodos que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción.

4. Widget → Es una aplicación pequeña que puede situarse en la pantalla de tu ordenador, web o móvil. En el caso de Android es una herramienta interactiva que tiene el objetivo de facilitar al usuario el acceso a ciertas funciones de la aplicación a la cual corresponde.

5. Gateway → También conocido como puerta de enlace. Es el dispositivo que actúa de interfaz de conexión entre aparatos o dispositivos, y también posibilita compartir recursos entre dos o más computadoras.

6. Backend → También conocido como capa de acceso a datos. Proporciona acceso simplificado a los datos almacenados y los modifica según esté programado.

7. Gap → En estadística, distancia o diferencia excesiva que existe entre elementos relacionados entre sí.

8. Outlier → En estadística es un elemento que es numéricamente distante del resto de los datos.

9. Framework → Es una estructura conceptual y tecnológica de asistencia definida con artefactos o módulos concretos que puede servir de base para la organización y desarrollo de software.

10. Protocolo → Es un sistema de reglas que permiten que dos o más entidades de un sistema de comunicaciones se comuniquen entre ellas para transmitir información por cualquier medio. Se trata de las reglas o estándar que define la sintaxis, semántica y sincronización de la comunicación, así como los posibles métodos de recuperación de los errores.

11. Layout → Es la estructura visual para una interfaz de usuario.

12. Clase → Es una plantilla para la creación de objetos de datos según un modelo predefinido. Se utilizan para representar entidades o conceptos.

13. Actividad → Cada una de las pantallas o vistas que forman una aplicación.

14. Servicio → Es un componente de una aplicación que puede realizar operaciones de larga ejecución en segundo plano y que no proporciona una interfaz de usuario.

15. Hub → En términos de programación es un contenedor que permite conectar entre si otros equipos o dispositivos retransmitiendo los paquetes de datos que recibe desde cualquiera de ellos hacia todos los demás.

CAPÍTULO 7: BIBLIOGRAFÍA

[1] Sensing & Control S.L. [Consulta: 23/10/2017] Disponible en: http://www.sensingcontrol.com/

[2] enControl web info [Consulta 23/10/2017] Disponible en: http://www.encontrol.io/

[3] enControl web *app* [Consulta 23/10/2017] Disponible en: http://encontrol.io/

[4] enControl Android *app* [Consulta 23/10/2017] Disponible en: https://play.google.com/store/apps/details?id=com.sensingcontrol.android

[5] enControl iOS *app* [Consulta 23/10/2017] Disponible en: https://itunes.apple.com/es/app/encontrol/id968651737?mt=8

[6] enControl vídeo explicativo [Consulta 4/12/2017] Disponible en: https://www.youtube.com/watch?v=qSYzRE9t-sw

[7] Azure [Consulta 19/12/2017] Disponible en: https://azure.microsoft.com

[8] Machine Learning [Consulta 19/12/2017] Disponible en: https://studio.azureml.net/

[9] Swagger [Consulta 19/12/2017] Disponible en: https://swagger.io/

[10] enControl API [Consulta 4/12/2017] Disponible en: https://encontrol.io/api/swagger-ui/

[11] enControl GitHub [Consulta 5/12/2017] Disponible en: https://github.com/Sensing-Control-DevTeam

[12] Raspberry Pi 3 [Consulta 19/12/2017] Disponible en: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[13] Razberry [Consulta 19/12/2017] Disponible en: https://razberry.z-wave.me/

[14] Z-Wave [Consulta 19/12/2017] Disponible en: https://z-wave.sigmadesigns.com/

[15] AppWidgetProviderInfo [Consulta 10/12/2017] Disponible en: https://developer.android.com/reference/android/appwidget/AppWidgetProviderInfo.html

[16] AppWidgetProvider [Consulta 10/12/2017] Disponible en: https://developer.android.com/reference/android/appwidget/AppWidgetProvider.html

[17] BroadcastReceiver [Consulta 10/12/2017] Disponible en: https://developer.android.com/reference/android/content/BroadcastReceiver.html

[18] Directrices de diseño para widgets en Android [Consulta 10/12/2017] Disponible en: https://developer.android.com/guide/practices/ui_guidelines/widget_design.html#anatomy_determining_size

[19] SignalR [Consulta 10/12/2017] http://signalr.net/

[20] Consulta sobre comportamiento inesperado de la función onUpdate() del App Widget Provider [Consulta 10/12/2017] https://stackoverflow.com/questions/9144262/android-widget-show-configuration-activity-before-widget-is-added-to-the-screen

[21] Incidencia presentada en Google sobre posible error en la documentación del AppWidgetProvider [Consulta 10/12/2017] https://issuetracker.google.com/issues/36924475#c0

[22] RemoteViews [Consulta 10/12/2017] Disponible en: https://developer.android.com/reference/android/widget/RemoteViews.html

[23] AppWidgetManager [Consulta 10/12/2017] Disponible en:https://developer.android.com/reference/android/appwidget/AppWidgetManager.html

[24] NUnit [Consulta 10/12/2017] Disponible en: http://nunit.org/

[25] JUnit [Consulta 10/12/2017] Disponible en: http://junit.org

[26] Servicios en Android [Consulta 20/12/2017] Disponible en: https://developer.android.com/guide/components/services.html?hl=es-419

[27] Atmosphere [Consulta 21/12/2017] Disponible en: https://github.com/Atmosphere/atmosphere

CAPÍTULO 8: ANEXOS

8.1 Manual de usuario

Para colocar un *widget* de enControl en la pantalla principal del móvil hay que seguir los siguientes pasos:

1. Instalar la aplicación enControl

2. Hacer *login* en la *app* con el usuario demo_uoc@demo_uoc.com y el *password* demo_uoc

3. Volver a la pantalla principal del teléfono y mantenerla pulsada durante 1 segundo aproximadamente hasta que aparezca el siguiente menú:

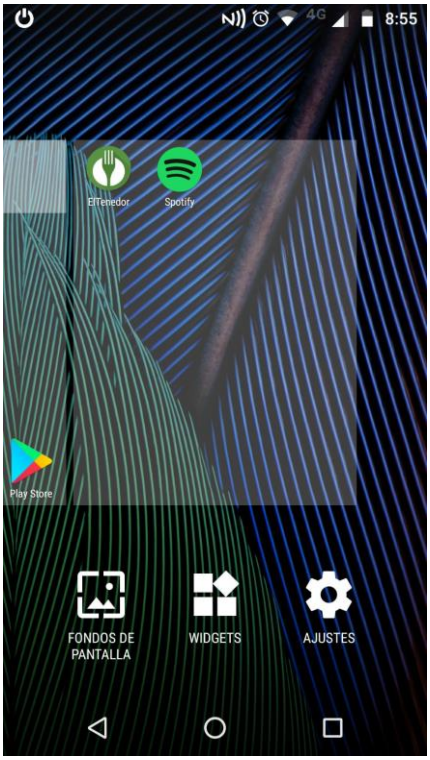

Fig. 8.1 Menú oculto

4. Presionar el apartado Widgets y buscar el correspondiente a enControl

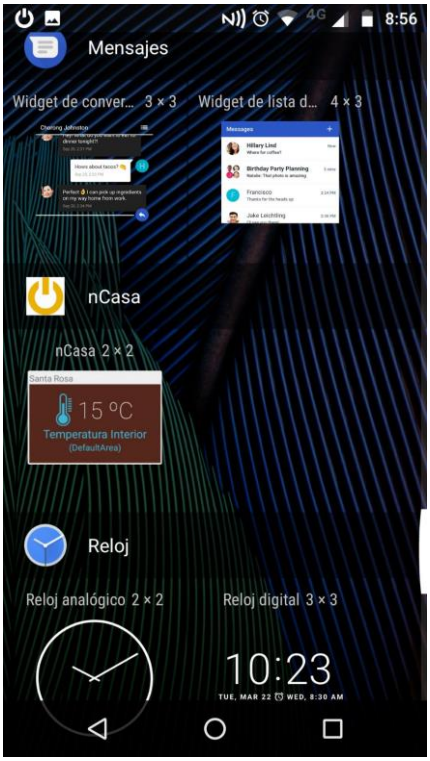

Fig. 8.2 Lista de *widgets*

5. Mantenerlo pulsado y arrastrarlo hasta la zona de la pantalla donde queramos ubicarlo

6. Seleccionar una instalación y seguidamente pulsar sobre el sensor deseado

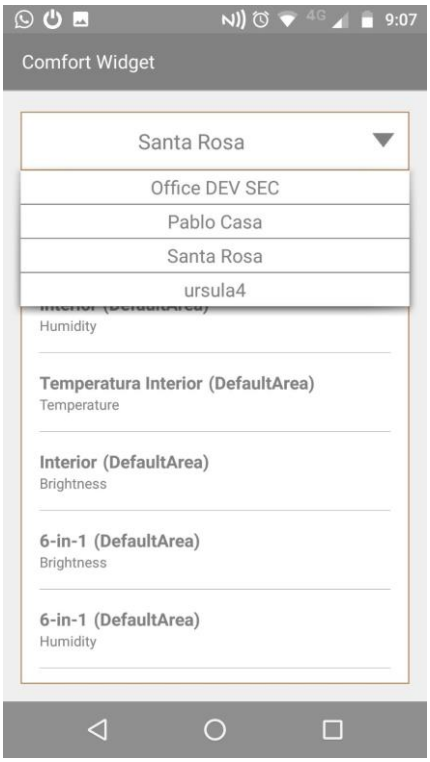

Fig. 8.3 Actividad de configuración

7. Después de esto ya tendremos el *widget* funcionando.

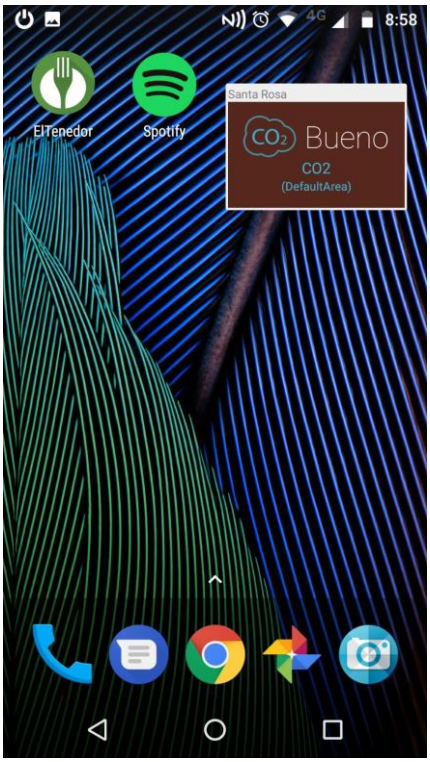

Fig. 8.4 *widget* colocado

8.2 Código del servidor

El software desarrollado en este trabajo se ha integrado dentro de la plataforma enControl de Sensing & Control S.L. El proyecto Android se ha librado entero en el .ZIP pertinente, pero el correspondiente al servidor no se puede entregar debido al secreto profesional de la empresa. Contiene muchísimo más código aparte del que se ha desarrollado aquí de vital importancia y gran valor. Sin embargo, en la figura 4.18 y 4.19 se puede ver la aportación a la API del servidor para este proyecto. A continuación se copian también los archivos completos referentes a los test NUnit.

```
using Microsoft.AspNet.SignalR.Hubs;
using nassist.Azure.BlobStorage;
using nassist.Azure.CloudTableStorage;
using nassist.ServiceImplementation.Services;
using nassist.ServiceImplementation.SignalR.Hubs;
using nassist.ServiceInterface.Services;
using nassist.ServiceModel;
using NSubstitute;
using NUnit.Framework;
using ServiceStack.OrmLite;
using System.Collections.Generic;
using System.Threading.Tasks;
using System.Linq;

namespace nassist.ServiceImplementation.Tests.InstallationTests
{
    [TestFixture]
    class InstallationServiceTests : ServiceTest
    {
        // Test service reference
        private IInstallationService installationService;

        [SetUp]
        protected override void SetUpBeforeEach()
        {
            base.SetUpBeforeEach();

            // Service dependencies (class properties and runtime resolved
services), keep them injected as singleton unless different substitutes needed
(by default)
            appHost.Container.Register<IBlobStorage>(c =>
Substitute.For<IBlobStorage>());
            appHost.Container.Register<ICloudTableStorage>(c =>
Substitute.For<ICloudTableStorage>());

appHost.Container.Register<IHubConnectionContext<IInstallationHubClient>>(c =>
Substitute.For<IHubConnectionContext<IInstallationHubClient>>());
            appHost.Container.Register<ISensorsService>(c =>
Substitute.For<ISensorsService>());
            appHost.Container.Register<ICameraService>(c =>
Substitute.For<ICameraService>());
            appHost.Container.Register<INodeService>(c =>
Substitute.For<INodeService>());
            appHost.Container.Register<IRuleService>(c =>
Substitute.For<IRuleService>());
            appHost.Container.Register<IGatewayService>(c =>
Substitute.For<IGatewayService>());
            appHost.Container.Register<ISchedulerService>(c =>
Substitute.For<ISchedulerService>());

            // Service under test
            appHost.Container.RegisterAutoWiredAs<InstallationService,
IInstallationService>();
            installationService =
appHost.Container.Resolve<IInstallationService>();
        }
```

Fig. 8.5 InstallationServiceTest.cs parte 1

```
[Test]
        public async Task GetComfortSensorsForInstallation()
        {
            List<SensorWithAreaName> comfortSensors = (await
installationService.Get(new InstallationSensorsComfort { Id =
testInstallation.Id })).Sensors;
            Assert.AreEqual(1, comfortSensors.Count());
            Assert.AreEqual(testSensor2, comfortSensors.Where(x => x.Id ==
testSensor2.Id.ToString()));
        }
    }
}
```

Fig. 8.6 InstallationServiceTest.cs parte 2

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Globalization;
using System.IO;
using System.Reflection;
using nassist.ServiceImplementation.BL.RequestVerification;
using nassist.ServiceModel;
using nassist.ServiceModel.Camera;
using NSubstitute;
using NUnit.Framework;
using ServiceStack;
using ServiceStack.Auth;
using ServiceStack.Caching;
using ServiceStack.Data;
using ServiceStack.OrmLite;
using ServiceStack.OrmLite.Sqlite;
using ServiceStack.OrmLite.SqlServer.Converters;
using ServiceStack.Testing;
using UserAuth = ServiceStack.Auth.UserAuth;

namespace nassist.ServiceImplementation.Tests
{
    public abstract class ServiceTest
    {
        protected ServiceStackHost appHost;
        private string ddlScript;

        // Some test data
        protected UserAuth testUser;
        protected Installation testInstallation;
        protected UserInstallation testUserInstallation;
        protected Floorplan testFloorplan;
        protected Area testArea;
        protected Node testNodeCam;
        protected Node testNodeSens;
        protected AreaNode testAreaNodeCam;
        protected AreaNode testAreaNodeSens;
        protected CameraNode testCameraNode;
        protected NodeSensor testNodeSensor;
        protected Sensor testSensor;
        protected Sensor testSensor2;
        protected Camera testCamera;
```

Fig. 8.7 ServiceTest.cs parte 1

```
[OneTimeSetUp]
        public void OneTimeSetUp()
        {
            var assembly = Assembly.GetExecutingAssembly();
            using (var stream =
assembly.GetManifestResourceStream(assembly.GetName().Name + ".ddl.sql"))
            {
                if (stream != null)
                {
                    using (StreamReader reader = new StreamReader(stream))
                    {
                        ddlScript = reader.ReadToEnd();
                    }
                }
            }

            appHost = new BasicAppHost().Init();
            var container = appHost.Container;

            // For the tests we use an inmemoty sqlite database so we can also
test the queries when needed without mocking DB
            var dialectProvider = new SqliteOrmLiteDialectProvider();
            dialectProvider.GetDateTimeConverter().DateStyle =
DateTimeKind.Utc;
            dialectProvider.RegisterConverter<DateTime>(new
DateTimeOffsetCustomConverter());
            dialectProvider.RegisterConverter<bool>(new
SqlServerBoolConverter());

            container.Register<IDbConnectionFactory>(new
OrmLiteConnectionFactory(":memory:", dialectProvider));
            container.Register<IAuthRepository>(c => new
OrmLiteAuthRepository(c.Resolve<IDbConnectionFactory>()));

            container.Register<ICacheClient>(new MemoryCacheClient {
FlushOnDispose = false });

            appHost.Config.AdminAuthSecret = ""; // To prevent authentication
errors with [Authenticate] or [RequireXXXRole] attributes if needed

            container.Register<AuthUserSession>(c => new AuthUserSession { Id
= "1", UserName = "test", UserAuthId = "1", IsAuthenticated = true, Roles =
new List<string> { "Admin" } });
            container.Register<IAuthSession>(c => new AuthUserSession { Id =
"1", UserName = "test", UserAuthId = "1", IsAuthenticated = true, Roles = new
List<string> { "Admin" } });

            // We are not testing veryfiers, so allow always to execute
            var requestVerifier = Substitute.For<IRequestVerifier>();
            requestVerifier.VerifyRequest(null, null).ReturnsForAnyArgs(true);
            container.Register<IRequestVerifier>(c => requestVerifier);
        }
```

Fig. 8.8 ServiceTest.cs parte 2

```
[OneTimeTearDown]
        public void OneTimeTearDown()
        {
            appHost.Dispose();
        }

        // Override this in your test class if you need different behaviour
        [SetUp]
        protected virtual void SetUpBeforeEach()
        {
            // Empty cache for every test
            appHost.Resolve<ICacheClient>().FlushAll();

            // Instantiating some test data
            testUser = new UserAuth { Id = 1, UserName = "admin", Email =
"admin@test.admin", CreatedDate = DateTime.Now, ModifiedDate = DateTime.Now };
            testInstallation = new Installation { Id = Guid.Parse("00000000-
0000-0000-0000-000000000001"), Type = "SensingControl", CreationDate =
DateTime.Now, LastModificationDate = DateTime.Now, ActivationDate =
DateTime.Now };
            testUserInstallation = new UserInstallation { Id = 1,
InstallationId = Guid.Parse("00000000-0000-0000-0000-000000000001"), UserId =
1, UserInGeofence = false };
            testFloorplan = new Floorplan { Id = 1, InstallationId =
Guid.Parse("00000000-0000-0000-0000-000000000001"), CreationDate =
DateTime.Now, LastModificationDate = DateTime.Now };
            testArea = new Area { Id = 1, Name = "DefaultArea", Description =
"Default Area", FloorplanId = 1, CreationDate = DateTime.Now,
LastModificationDate = DateTime.Now };
            testNodeCam = new Node { Id = 1, Name = "Test Node Cam",
Description = "Test Node Cam", CreationDate = DateTime.Now,
LastModificationDate = DateTime.Now };
            testNodeSens = new Node { Id = 2, Name = "Test Node Sens",
Description = "Test Node Sens", CreationDate = DateTime.Now,
LastModificationDate = DateTime.Now };
            testAreaNodeCam = new AreaNode { AreaId = 1, NodeId = 1 };
            testAreaNodeSens = new AreaNode { AreaId = 1, NodeId = 2 };
            testCameraNode = new CameraNode { CameraId = "00000000-0000-0000-
0000-000000000001", NodeId = 1 };
            testNodeSensor = new NodeSensor { SensorId = "00000000-0000-0000-
0000-000000000001", NodeId = 2 };
            testNodeSensor = new NodeSensor { SensorId = "00000000-0000-0000-
0000-000000000002", NodeId = 2 };

            testSensor = new Sensor
            {
                Id = "00000000-0000-0000-0000-000000000001",
                Type = SensorType.EnergyMeter.ToString(),
                Name = "Test",
                Value = 5,
                Status = "ok",
                LastDateValue = DateTime.UtcNow,
                LastDateStatus = DateTime.UtcNow,
                Configuration = "",
                IsVirtual = false,
                DeviceStatus = DeviceStatus.ok.ToString(),
                DeviceStatusDate = DateTime.UtcNow
            };
```

Fig. 8.9 ServiceTest.cs parte 3

```
testSensor2 = new Sensor
            {
                Id = "00000000-0000-0000-0000-000000000002",
                Type = SensorType.Temperature.ToString(),
                Name = "Test",
                Value = 20,
                Status = "ok",
                LastDateValue = DateTime.UtcNow,
                LastDateStatus = DateTime.UtcNow,
                Configuration = "",
                IsVirtual = false,
                DeviceStatus = DeviceStatus.ok.ToString(),
                DeviceStatusDate = DateTime.UtcNow
            };

            testCamera = new Camera
            {
                Id = "00000000-0000-0000-0000-000000000001",
                Name = "test",
                Type = "Remote",
                Manufacturer = "foscam",
                Model = "FI9821P",
                Description = "Test Camera",
                IpAddress = "192.168.1.120",
                Status = CameraStatusType.ok.ToString(),
                LastDateStatus = DateTime.Now,
                Port = 88,
                LastDateImage = DateTime.Now,
                LastImageName =
"https://sensingcloudstoragedev.blob.core.windows.net/photos/81669e23-c239-
4b8b-a063-2d6c818dbe4e/Device/20161229204822582000.jpg",
                LastSensorId = "8bc19e82-7275-4ce1-9fea-add835629dff",
                Configuration =
"{\"VideoURL\":\"/videoMain\",\"VideoFormat\":\"MPEG\",\"VideoProtocol\":\"rts
p\",\"RtspPort\":\"88\",\"PictureURL\":\"/cgi-
bin/CGIProxy.fcgi?usr={user}&pwd={pwd}&cmd=snapPicture2\",\"PictureFormat\":\"
JPEG\",\"PictureProtocol\":\"http\",\"ZoomInURL\":null,\"ZoomOutURL\":null,\"P
anLeftURL\":null,\"PanRightURL\":null,\"TiltUpURL\":null,\"TiltDownURL\":null,
\"PTZStopURL\":null,\"Authentication\":null,\"User\":\"administrator\",\"Passw
ord\":\"32prov511\"}",
                CreationDate = DateTime.Now
            };
```

Fig. 8.10 ServiceTest.cs parte 4

```
// Create required tables for the tests and fill them with test data (if
needed)
            // This is recreated for every test using an in-memory sqlite
database (easier and faster than mocking db connection)
            using (var db = appHost.Resolve<IDbConnectionFactory>().Open())
            {
                // We don't strictly follow SStack DB primary key design
conventions so we must create DB tables via script rather than OrmLite methods
                db.ExecuteSql(ddlScript);
                db.Insert(testUser);
                db.Insert(testInstallation);
                db.Insert(testUserInstallation);
                db.Insert(testFloorplan);
                db.Insert(testArea);
                db.Insert(testNodeSens);
                db.Insert(testNodeCam);
                db.Insert(testAreaNodeCam);
                db.Insert(testAreaNodeSens);
                db.Insert(testCamera);
                db.Insert(testSensor);
                db.Insert(testSensor2);
                db.Insert(testCameraNode);
                db.Insert(testNodeSensor);
            }
        }

        private class DateTimeOffsetCustomConverter : OrmLiteConverter
        {
            public override string ColumnDefinition
            { get { return "DATETIMEOFFSET"; } }

            public override DbType DbType
            { get { return DbType.DateTimeOffset; }

            //From OrmLiteDialectProviderBase:
            public override object FromDbValue(Type fieldType, object value)
            {
                var strValue = value as string;
                if (strValue != null)
                {
                    return DateTimeOffset.Parse(strValue, null,
DateTimeStyles.RoundtripKind).DateTime;
                }

                if (value is DateTime)
                { return (DateTime)value; }

                if (value is DateTimeOffset)
                {
                    return ((DateTimeOffset)value).DateTime;
                }

                return
DialectProvider.StringSerializer.DeserializeFromString(value.ToString(),
fieldType);
            }
        }
    }
}
```

Fig. 8.11 ServiceTest.cs parte 5