FACULTAD DE INGENIERÍA Y CIENCIAS AGROPECUARIAS

DESARROLLO E IMPLANTACIÓN DE APLICACIÓN MÓVIL PARA EL CONTROL Y MONITORIZACIÓN DE CALENTADOR DE AGUA DE ACUMULACIÓN ELÉCTRICO

Trabajo de Titulación presentado en conformidad con los requisitos establecidos para optar por el título de Ingeniero en Sistemas de Computación e Informática

Profesor Guía
MSc. Paulo Roberto Guerra Terán

Autor
Rafael Alejandro Kelly Plua

Año
2016

DECLARACIÓN DEL PROFESOR GUÍA

“Declaro haber dirigido este trabajo a través de reuniones periódicas con el estudiante, orientando sus conocimientos y competencias para un eficiente desarrollo del tema escogido y dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación.”

Paulo Roberto Guerra Terán
MSc. en Software y Sistemas
C.I: 1002856050

DECLARACIÓN DE AUTORÍA DEL ESTUDIANTE

“Declaro que este trabajo es original, de mi autoría, que se han citado las fuentes correspondientes y que en su ejecución se respetaron las disposiciones legales que protegen los derechos de autor vigentes.”

Rafael Alejandro Kelly Plua
C.I.: 1716024714

RESUMEN

El interés por utilizar elementos tecnológicos para la automatización del hogar, principalmente para el confort de quienes en él habitan, no es una tendencia reciente. Sin embargo, lo que sí destaca en la actualidad es la disponibilidad de herramientas ideales para plasmar dicho interés en soluciones concretas. Incluso, los conceptos alguna vez limitados al hogar han expandido su alcance, construyendo el popular término el Internet de las cosas. Es así que el presente proyecto de titulación tiene por objetivo el desarrollo de un aplicativo móvil para el control y monitorización de un calentador de agua de acumulación eléctrico. Para esto, se analizan algunos de los principales protocolos de comunicación útiles para el Internet de las cosas, para luego aplicar el más adecuado al problema planteado. Como resultado, se establece a *MQ Telemetry Transport* como el protocolo que mejor se acopla a las necesidades del proyecto. Dicho resultado pone en evidencia la inexistencia de un único mecanismo de comunicación que pueda responder a todas las posibles necesidades que un sistema destinado al Internet de las cosas pueda tener.

ABSTRACT

Applying technology in home automation, with a focus on the comfort of its residents, is not a recent trend. However, the availability of tools that are ideal for this purpose is something that stands out nowadays. Concepts once limited only to the domestic environment have even been given a broader use, resulting in the Internet of things, a fairly popular term in recent times. Taking that into account, the goal of the following project is to develop a mobile app for controlling and monitoring an electric water heater. To achieve this, some of the main Internet of things protocols are evaluated, to later determine which one is the most appropriate for the given problem. As a result, *MQ Telemetry Transport* was chosen to accomplish the goal of the project. This result makes it clear that there is not a single communication strategy that's is able to satisfy the broad range of applications the Internet of thing can have.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. Introducción

1.1. Antecedentes

El término domótica es considerado como un sinónimo de automatización del hogar. Este último es definido como el campo dentro de la automatización de construcciones, especializado en los requerimientos específicos de casas privadas y en la aplicación de técnicas de automatismo para el confort y seguridad de sus residentes. (Carbou, Diaz, Exposito & Roman, 2013)

Tecnologías para el desarrollo de soluciones en el campo de la domótica han existido desde hace varias décadas. Por ejemplo, el protocolo X10 diseñado para el control de dispositivos/electrodomésticos, utilizando una línea de poder como canal de comunicación, fue diseñado en 1975. (Carbou et al., 2013) Sin embargo, el rápido avance en las comunicaciones móviles en los últimos años ha introducido, de igual forma, un avance tecnológico representativo en la automatización del hogar. Redes wireless (3G, 4G, Wi-Fi) y dispositivos inteligentes, con interfaces de comunicación wireless (Bluetooth, Zigbee, Wi-Fi), son omnipresentes, y permiten al usuario llevar el control del hogar y la automatización de edificaciones al siguiente nivel. En vez de funcionalidades rudimentarias, la automatización del hogar hoy en día puede entregar capacidades con un impacto real en el confort, seguridad y conservación de energía en espacios residenciales e industriales. (Kyas, 2013)
Similar importancia han tenido los avances en la interfaz de usuario. La revolución de los smartphones y tablets ha logrado la presencia en el hogar de dispositivos que pueden cumplir la función de controles remotos personales y universales. Paneles estacionarios y dispositivos de control propietarios han perdido relevancia siendo reemplazados por apps, que son fáciles de operar, mantener y actualizar. (Kyas, 2013)

Tomando en consideración lo expuesto previamente, se propone el desarrollo e implementación de una solución de domótica enfocada al control y monitoreo de

un calentador de agua de acumulación eléctrico, empleando como interfaz una aplicación móvil.

1.2. Alcance

El alcance del trabajo de titulación a realizarse es analizar, desarrollar e implantar una solución de domótica para el control y monitoreo de un calentador de agua de acumulación eléctrico.

La interfaz de usuario será una aplicación móvil que permitirá al usuario, como funcionalidad más básica, el encendido y apagado remoto del dispositivo calentador. El usuario será capaz de establecer eventos de encendido (basados en hora del día), con posibilidad de fijar repetición de los mismos (basada en los días de la semana) o realizar su cancelación. La aplicación permitirá visualizar la temperatura del agua almacenada y notificará el estado del calentador (encendido o apagado), para los eventos programados. Adicionalmente se registrará y desplegará información del consumo eléctrico generado por el dispositivo.

Para lograr esta interacción con el calentador de agua, se utilizará un microcontrolador Arduino con las adecuaciones necesarias para proveer la funcionalidad descrita.

La realización del proyecto representa una aplicación de conocimientos adquiridos en materias de desarrollo de software, así como una extrapolación de lo aprendido en tema de microcontroladores. Como metodología específica de la carrera, se pretende utilizar ciertas prácticas prescritas en *Scrum*.

1.3. Justificación

Con la realización del proyecto se pretende tener impacto sobre dos aspectos mencionados en los antecedentes: el confort y, principalmente, el ahorro de energía eléctrica.

Un calentador de agua eléctrico de acumulación requiere ser encendido con anticipación para que el líquido alcance la temperatura que el usuario desea. El manejo manual de este proceso puede llegar a ser, en primera instancia, molesto y fundamentalmente impreciso en el control del tiempo que el dispositivo permanece encendido. El mecanismo propuesto proveerá exactitud en este último aspecto, así como información veraz sobre el impacto del uso del dispositivo en el consumo eléctrico del hogar. Además, empleará una plataforma, dispositivos móviles, cuyo uso se ha extendido considerablemente en tiempos recientes, lo que facilita la integración del aplicativo a la rutina del usuario.

Existen varios dispositivos disponibles para la automatización del hogar. Sin embargo, muchos de los sistemas comerciales disponibles tienen un precio elevado, no pueden ser adaptados a las necesidades del usuario y utilizan interfaces de usuario poco actuales. Por otra parte, gracias a la flexibilidad de la plataforma Arduino es posible interactuar con facilidad con varios sensores y actuadores, lo que permite la implementación de soluciones de automatización.

Finalmente, dentro del contexto nacional, la eliminación del subsidio del gas licuado de petróleo vuelve al uso de energía eléctrica para el calentamiento de agua una alternativa atractiva, pero sobre la cual se requiere tener un control adecuado.

1.4. Objetivo General

Desarrollar e implantar una solución de domótica enfocada al control y monitoreo de un calentador de agua de acumulación eléctrico.

1.5. Objetivos Específicos

- Analizar los principales protocolos de conectividad y estilos para exponer remotamente un servicio de software destinado para una solución del Internet de las cosas.
- Desarrollar un API, basándose en el análisis previo, para exponer la funcionalidad de interacción con el dispositivo calentador de agua, así como para la provisión de datos resultantes del monitoreo.
- Desarrollar una aplicación móvil a emplearse como interfaz de usuario de la solución.

1.6. Metodología a Utilizar

Para el desarrollo del proyecto se empleará el método experimental, cuya forma de abordar el proceso se caracteriza, en gran medida, por la identificación de conceptos que facilitan soluciones a un problema y la posterior evaluación de soluciones mediante la construcción de sistemas prototipo (Dodig-Crnkovic, s. f.). Tanto para la implementación de componentes de hardware, así como para los elementos de software esenciales para el cumplimiento del objetivo, esta metodología permitirá partir de la investigación bibliográfica detallada hasta llegar a la aplicación práctica de dicha información. Este proceso experimental proveerá de resultados tangibles útiles para validar el cumplimiento del propósito del proyecto.

2. Marco Teórico

2.1. Internet de las Cosas

En los antecedentes del proyecto se definió el término domótica. No obstante, se ha llevado las ideas originalmente pensadas para el hogar un paso más allá con el objetivo de alcanzar un mundo de objetos interconectados y conectados a la red, capaces de cumplir cada vez más funciones y más complejas: el Internet de las cosas (Aparicio, 2015). McEwen y Cassimally (2013, p. 11) plantean la ecuación de la Figura 1 para definir, de manera sencilla, a esta expresión.

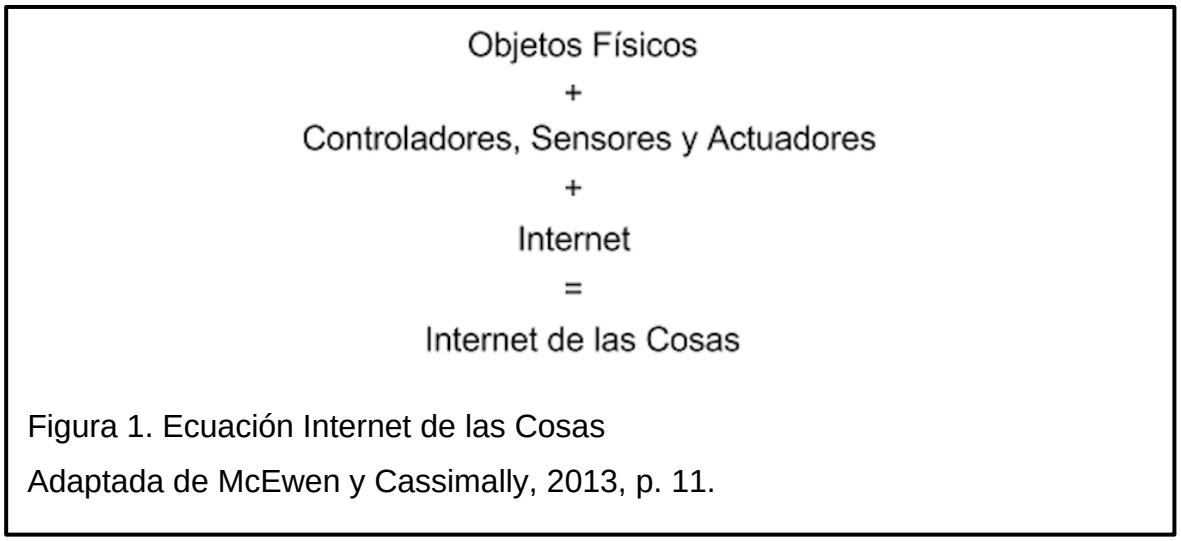


Figura 1. Ecuación Internet de las Cosas
Adaptada de McEwen y Cassimally, 2013, p. 11.

Un sensor es un dispositivo que detecta eventos y lee cambios en su ambiente. Un actuador ejecuta acciones en el mundo físico, basadas en comandos típicamente recibidos del Internet. Un controlador es un elemento que provee o contiene la lógica de la aplicación del Internet de las cosas. (Waher, 2015, p. 11)

En lo que resta del documento se utilizará el término Internet de las cosas. Los conceptos expuestos son aplicables tanto al Internet de las cosas como a la domótica.

2.2. Protocolos de Comunicación

2.2.1. HTTP

HTTP (*Hypertext Transfer Protocol*) es un protocolo petición/respuesta en el que un cliente solicita información de un servidor, el cual responde a dichas solicitudes (Waher, 2015, p. 36). Como se especifica en el diagrama de secuencia de la Figura 2, una petición se compone básicamente de un método, un recurso, algunos encabezados y contenido opcional. Por su parte, una respuesta está formada por un código de estado de tres dígitos, un grupo de encabezados y contenido opcional.

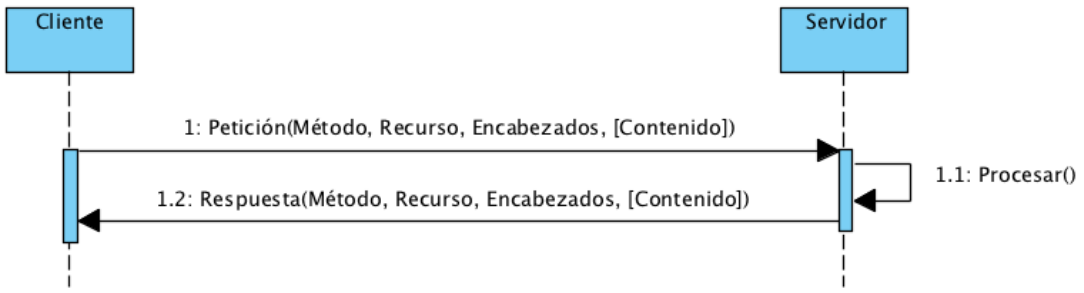


Figura 2. Patrón Petición/Respuesta HTTP
Adaptada de Waher, 2015, p. 36.

De los métodos dependerá la acción realizada por el servidor. Los métodos más comunes son: *GET* usado para recuperar información del servidor; *POST* para enviar contenido al servidor para ser procesado; *PUT* para almacenar el contenido de la solicitud; y *DELETE* para remover recursos.

Un recurso, originalmente pensado como una colección de documentos de hipertexto (documentos HTML), es identificado por un localizador uniforme de recursos o URL (*Uniform Resource Locator*) por sus siglas en inglés. Un URL está compuesto de cinco partes:

- Primero, el esquema URI (*Uniform Resource Identifier*) que describe la forma en la que se accederá al recurso.
- La autoridad, que identifica al servidor que acoge al recurso. Puede ser un nombre de dominio o una dirección IP, seguida opcionalmente por un número de puerto.
- La ruta que identifica al recurso como tal, en el contexto de la autoridad a la que se envía la petición.
- Por último, parámetros de consulta (*query parameters*) y un identificador de fragmento. Ambos elementos son opcionales.

La Figura 3 provee un ejemplo de las distintas partes de un URL.

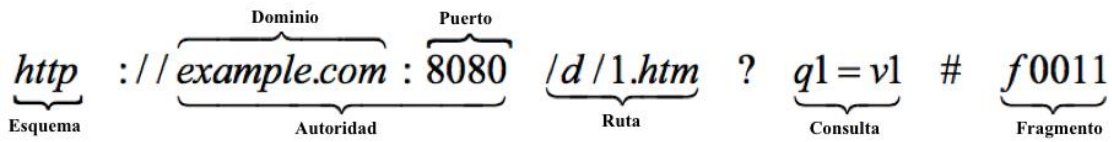


Figura 3. Estructura de un URL
Adaptada de Waher, 2015, p. 36.

HTTP define un conjunto de encabezados, pares de texto llave/valor, que pueden ser utilizados para agregar *metadata* acerca de las solicitudes y contestaciones enviadas en la red. Entre la información adicional que puede ser expresada en estos encabezados se encuentra la codificación del contenido, su longitud, el tipo de contenido deseado, etc. Adicionalmente, también proveen formas básicas de autenticar clientes en el servidor y *cookies*, un mecanismo para introducir estado al protocolo.

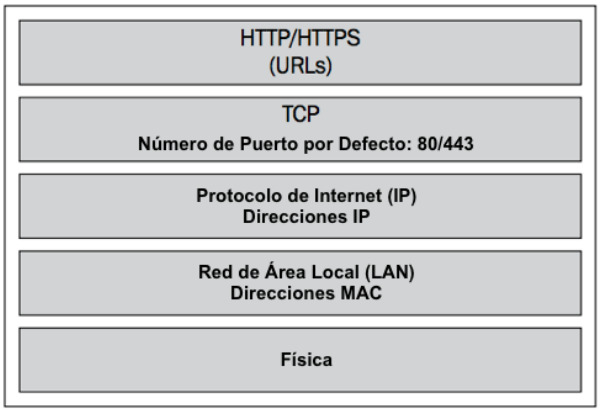


Figura 4. Arquitectura HTTP
Adaptada de Waher, 2015, p. 37.

La Figura 4 describe la arquitectura del protocolo HTTP. Este protocolo trabaja sobre IP (*Internet Protocol*), en el que la identificación de dispositivos se realiza empleando una dirección IP que posibilita la comunicación entre distintas LANs (áreas de redes locales). La comunicación se realiza sobre una conexión TCP (*Transmission Control Protocol*) entre el cliente y el servidor. La conexión TCP se asegura de que no haya pérdida de paquetes y que estos sean recibidos en el mismo orden en el que fueron enviados. Los extremos de la conexión (*endpoints*) son definidos por la combinación de la dirección IP y un puerto.

Waher (2015, p. 38) señala que, en el contexto del Internet de las cosas, existen tres estrategias básicas que se pueden elegir para publicar datos utilizando HTTP:

1. El sensor actúa como un cliente que publica información a un servidor en el Internet. El servidor actúa como un *broker* e informa a las partes interesadas acerca de los valores del sensor. Este patrón se denomina publicación/suscripción y será tratado con mayor detalle posteriormente. Provee la ventaja de simplificar el manejo de eventos, pero se dificulta la obtención de valores momentáneos. Los sensores también pueden ser

colocados detrás de un firewall, siempre y cuando el servidor esté públicamente disponible en el Internet.

2. Dejar que todas las entidades en la red actúen como clientes y servidores, dependiendo de qué necesiten hacer. Este patrón se discutirá en la siguiente sección número 2.2.1.1. Se reduce la latencia en la comunicación pero se requiere que todos los participantes se encuentren en el mismo lado del firewall.

3. Contrario a la primera opción, el sensor actúa como servidor y cualquiera que esté interesado en conocer su estado deberá convertirse en un cliente. Esto trae ventajas al acceder a valores momentáneos, pero el envío de eventos se torna más difícil. Otra de las ventajas es que permite fácil acceso al sensor por parte de participantes conectándose detrás de firewalls, siempre y cuando el sensor esté públicamente disponible.

Es importante recalcar que, si bien en las opciones anteriores se introduce el concepto de eventos, la naturaleza de petición/respuesta del protocolo representa una limitación considerable para el soporte de ese tipo de funcionalidad. HTTP no ofrece un mecanismo por defecto para conocer cuándo ocurrirá algún evento cuya realización se pueda estar esperando, así que no es posible realizar una solicitud que coincida exactamente con el momento en el que la información se vuelva disponible. McEwen y Cassimally (2013, p. 199) indican que la forma tradicional de manejar estos casos es la ejecución de peticiones HTTP en intervalos regulares; esta técnica es conocida como sondeo o *polling*. Lograr mantener a los clientes sincronizados con los datos del servidor requiere que el sondeo se realice con una regularidad alta (por ejemplo, cada 10 segundos). Esto último aumenta la carga que el servidor debe soportar e incluso tiene efectos negativos en clientes que, debido a sus limitados recursos de procesamiento, bloquean sus actividades hasta obtener una respuesta (el caso de varios microcontroladores). Como alternativa a las ineficiencias de *polling*, un

grupo de tecnologías englobadas bajo el nombre de *Comet* fueron desarrolladas (McEwen y Cassimally, 2013, p. 199):

- *Long Polling* (unidireccional).
  Se inicia con una petición de sondeo inicial, por parte del cliente. Sin embargo, el servidor no envía la respuesta final ni cierra la conexión inmediatamente y, en cambio, espera hasta que haya nueva información disponible. Esto significa que el servidor debe enviar regularmente mensajes *keep-alive* que, como su nombre sugiere, tienen el propósito de que la conexión se mantenga activa. Aunque se pueden obtener datos en tiempo real, todavía comparte las desventajas del *polling* tradicional.

- *Multipart XMLHttpRequest* (unidireccional).
  Al construir aplicaciones web es muy común utilizar un API JavaScript llamado *XMLHttpRequest* para comunicarse con un servidor web sin la necesidad de recargar una página en su totalidad. Tiene similitudes con *long polling*, utilizando un tipo de contenido conocido como *multipart/x-mixed-replace* con lo cual es posible que el servidor envíe subsecuentes versiones de un documento, una vez que la conexión inicial con un cliente se haya establecido.

Por último, otra alternativa es optar por *Websockets* HTML5, un protocolo que emplea una única conexión TCP sobre la cual se puede enviar tráfico bidireccionalmente. *Websocket* es un protocolo independiente basado en TCP, relacionado con HTTP debido a que la solicitud de apertura (*handshake*), enviada por el cliente, es interpretada por servidores HTTP como una petición de actualización (Fette y Melnikov, 2011).

2.2.1.1. UPnP

UPnP (*Universal Plug and Play*) es un estándar que ayuda a dispositivos en redes IP *ad hoc* a descubrirse, detectar los servicios alojados en cada dispositivo, ejecutar acciones y reportar eventos (Waher, 2015, p. 65). Una red *ad hoc* se caracteriza por no tener una configuración o topología definida. En una de estas redes, los dispositivos se encuentran entre ellos y se adaptan al entorno que los rodea.

UPnP está primariamente basado en una aplicación de HTTP en la que los participantes desempeñan tanto el rol de cliente como el de servidor. El inconveniente de utilizar HTTP para encontrar nuevos recursos es que trabaja sobre TCP, y es así que se realiza sobre conexiones entre dos extremos que deben conocerse. UDP (*User Datagram Protocol*) no presenta esta limitación y, por esta razón, se lo utiliza para extender el protocolo proveyendo direccionamiento *unicast* (HTTPU) y *multicast* (HTTPMU). El descubrimiento de dispositivos se realiza por medio de SSDP (*Simple Service Discovery Protocol*), basado en HTTP sobre UDP, y la suscripción y notificación de eventos se basa en GENA (*General Event Notification Architecture*). SSDP y GENA introducen nuevos métodos HTTP para buscar, notificar, suscribirse y cancelar la suscripción a un evento. Los dispositivos se encuentran unos a otros notificando de su existencia usando direccionamiento *multicast*. Usando dicho direccionamiento, también pueden buscar ciertos tipos de dispositivos y servicios. Acciones en servicios son invocadas empleando llamadas de servicios web SOAP (*Simple Object Access Protocol*).

Una consideración importante de usar HTTP sobre UDP es que no existe soporte para conexiones. Por lo tanto, la entrega de los datagramas, y por consiguiente su orden, no está garantizada. Esto hace que enviar peticiones o respuestas grandes no sea trivial.

2.2.2. CoAP

Waher (2015, p. 98) indica que UPnP representa los beneficios de simplificar HTTP y utilizarlo sobre UDP en vez de TCP. No obstante, también recalca que para dispositivos y redes IP con recursos extremadamente limitados HTTPU no es una opción práctica por el alto consumo de banda y recursos. Esto último es particularmente cierto cuando la red subyacente limita el tamaño de los datagramas, como es el caso de usar IPv6 sobre 6LoWPAN (*Low Power Wireless Personal Area Networks*). CoAP (*Constrained Application Protocol*) es un intento por resolver esto.

La principal diferencia entre CoAP y HTTPU es que CoAP reemplaza los encabezados de texto usados en HTTPU con encabezados binarios más compactos y, por consiguiente, esto reduce el número de opciones disponibles en ellos. Esto hace que sea mucho más fácil codificar y decodificar mensajes CoAP. CoAP también reduce el número de métodos utilizados a solo 4: *GET*, *POST*, *PUT* y *DELETE*.

La principal limitación de CoAP es que al ser relativamente nuevo (su especificación fue publicada en Junio de 2014) el conjunto de herramientas de desarrollo disponibles es un tanto reducido.

2.2.3. MQTT

MQTT (*MQ Telemetry Transport*) es un ligero protocolo de comunicación cliente servidor de publicación/suscripción (Banks y Gupta, 2014).

El patrón de publicación/suscripción se caracteriza por permitir el desacoplamiento entre el proveedor de información, llamado el publicador, y los consumidores de dicha información, conocidos como suscriptores (Lampkin, V. et al., 2012, p. 26). Esto se logra con la introducción de un *broker* de mensajes como actor intermedio, que se responsabiliza de la recepción de contenido y su

transmisión hacia los suscriptores interesados. Frente al patrón tradicional punto a punto, la ventaja del modelo de publicación/suscripción es que el dispositivo o aplicación publicadora no necesita conocer nada acerca de los suscriptores, y viceversa. El publicador simplemente envía el mensaje y en su contenido incluye un identificador que denota su tópico o ámbito. El *broker* distribuirá el mensaje a todas las aplicaciones o dispositivos que hayan optado por suscribirse a ese tópico. De esta forma, en vez de enviar información a una dirección específica de destino, se logra una comunicación *multicast* basada en contenido.

En MQTT, los tópicos se encuentran organizados jerárquicamente en un árbol. Como se puede apreciar en la Figura 5, el identificador de un tópico es una cadena de caracteres, en la que se emplea el carácter / (*slash*) como delimitador de niveles al momento de expresar una ruta (similar a un sistema de archivos).

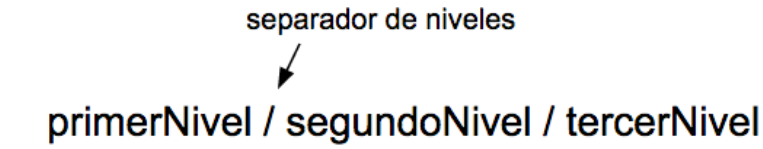


Figura 5. Identificador de Tópico

Un identificador de tópico puede ser explícito o contener alguno de los dos caracteres comodín válidos (Figura 6): + para sustituir un único nivel o # para cubrir un número arbitrario de niveles (una rama).

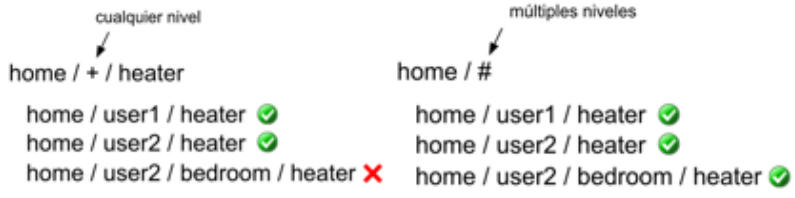


Figura 6. Comodines para Identificador de Tópico

En el diagrama de la Figura 7 se muestra la arquitectura del protocolo. Comparándola con la arquitectura de HTTP (Figura 4), se puede apreciar que

también se ejecuta sobre TCP/IP; sin embargo, a diferencia de HTTP, los puertos reservados para MQTT son el 1883 y 8883.

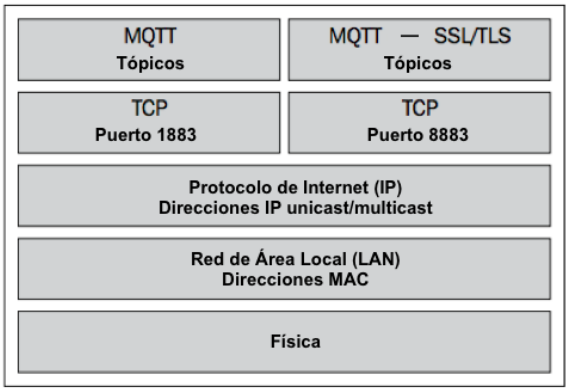


Figura 7. Arquitectura Protocolo MQTT
Adaptada de Waher, 2015, p. 109.

De acuerdo a Lampkin et. al. (2012, p. 6) otras características del protocolo incluyen:

- Transporte agnóstico al contenido.
  HTTP, por ejemplo, se centra en el documento y soporta el estándar MIME (*Multipurpose Internet Mail Extensions*) para definir tipos de contenidos. En cambio, al usar MQTT los datos simplemente se transmiten como un arreglo de bytes.

- Ofrece tres niveles de calidad de servicio en la publicación de mensajes:
  - QoS 0 – como máximo una vez: donde los mensajes son entregados de acuerdo a los mejores esfuerzos del sistema de operaciones y pueden existir pérdidas.
  - QoS 1 – por lo menos una vez: donde se garantiza que los mensajes arribarán pero duplicaciones pueden ocurrir. Cada destinatario confirma la recepción de la información publicada.
  - QoS 2 – exactamente una vez.

- Costo reducido de transporte.
  El encabezado de MQTT es corto y el tamaño de paquete más pequeño posible es de 2 bytes. La especificación es breve y existe un número limitado de tipos de mensaje (*CONNECT*, *PUBLISH*, *SUBSCRIBE*, *UNSUBSCRIBE*, y *DISCONNECT*).

- Mensajes retenidos.
  El servidor tiene la capacidad de almacenar mensajes aún después de que han sido enviados a todos los suscriptores actuales. Si una nueva suscripción es recibida para el mismo tópico, cualquier mensaje retenido será enviado al nuevo cliente.

- Sesiones limpias y conexiones duraderas.
  Cuando un cliente se conecta al *broker* se almacena una bandera de sesión limpia. Si dicha bandera tiene un valor verdadero, todas las suscripciones del cliente serán removidas cuando se desconecte del servidor. Caso contrario, la conexión es tratada como duradera y las suscripciones persistirán aún luego de una desconexión. En este último caso, mensajes subsecuentes que arriben con un QoS mayor a cero serán almacenados y transmitidos al reestablecerse la comunicación.

- Voluntades.
  Al momento de conectarse, un cliente puede informarle al servidor que tiene un mensaje que deberá ser publicado a uno o varios tópicos específicos en el evento de una pérdida de conexión inesperada.

2.2.3.1. Amazon IoT

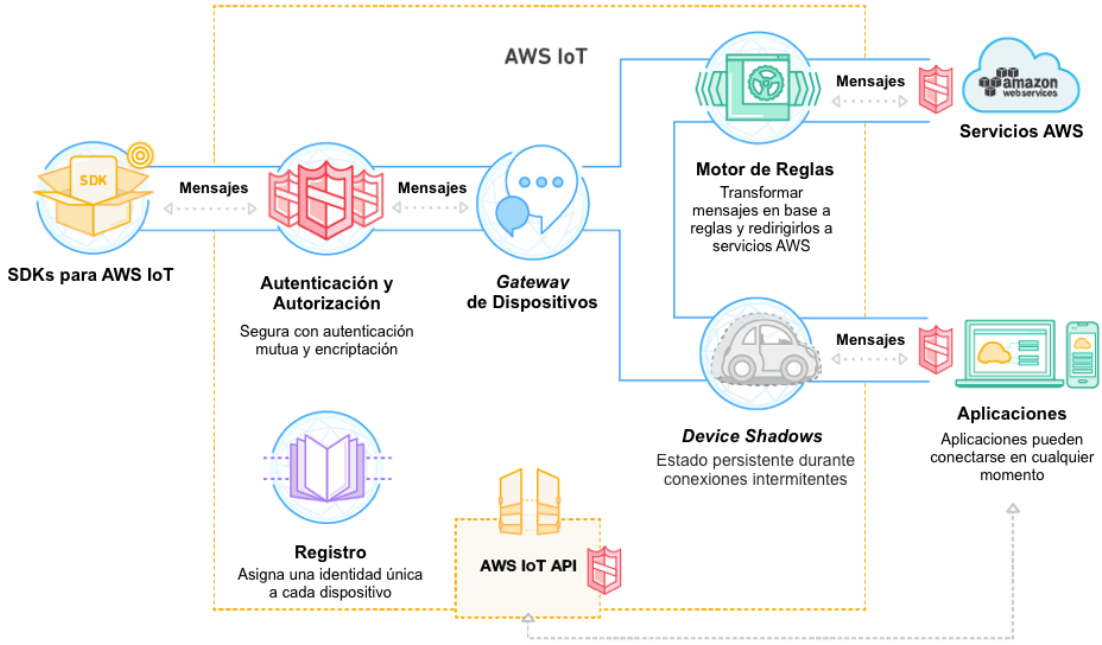


Figura 8. Componentes de Amazon IoT
Adaptada de “AWS IoT Developer Guide”, s.f.

Amazon IoT forma parte del conjunto de servicios web provistos por esta compañía, conocidos comúnmente por su acrónimo AWS (*Amazon Web Services*). En la Figura 8 se puede visualizar sus componentes y como se relacionan unos con otros. Los elementos más relevantes son los siguientes (“AWS IoT Developer Guide”, s.f.):

- Broker de mensajes.
  Corresponde a lo que en la Figura 8 se encuentra rotulado como *Gateway* de Dispositivos. Soporta el uso de MQTT para publicación y suscripción y ofrece capacidades de publicación mediante una interfaz REST (*Representational State Transfer*) sobre HTTPS (*Hypertext Transfer Protocol Secure*). Adicionalmente, ofrece la opción de utilizar MQTT sobre el protocolo WebSocket. Si bien es primariamente una implementación de

la versión 3.1.1 de MQTT, presenta las siguientes particularidades respecto a la especificación:

- Suscribirse a un tópico con calidad de servicio nivel 0 significará que un mensaje será entregado cero o más veces (podrá existir duplicidad en la entrega). No se ofrece soporte para QoS 2.
- No existe soporte para mensajes retenidos, así como tampoco para sesiones persistentes. Es así que todas las sesiones son limpias.

- Servicio de *Thing Shadows*.

  Provee representaciones persistentes de cosas en la nube de AWS (aplicaciones o dispositivos denominados cosas en la terminología de la plataforma debido a su uso en el Internet de las cosas). Un *device shadow* es un documento JSON (*JavaScript Object Notation*) usado para almacenar y recuperar el estado actual de estos dispositivos. En el ejemplo mostrado a continuación (Figura 9) se puede observar su estructura:

```
{
    "state" : {
        "desired" : {
          "color" : "RED",
          "sequence" : [ "RED", "GREEN", "BLUE" ]
        },
        "reported" : {
          "color" : "GREEN"
        }
    },
    "metadata" : {
        "desired" : {
            "color" : {
                "timestamp" : 12345
            },
            "sequence" : {
                "timestamp" : 12345
            }
        },
        "reported" : {
            "color" : {
                "timestamp" : 12345
            }
        }
    },
    "version" : 10,
    "clientToken" : "UniqueClientToken",
    "timestamp": 123456789
}
```


Figura 9. *Device Shadow*

En sus atributos se almacena:

1. El estado, tanto deseado como reportado.
2. *Metadata* sobre los atributos que componen el estado indicando la última actualización mediante un *timestamp*.
3. La versión del documento, incrementada con cada actualización y utilizada para asegurar que el documento sea el más reciente al realizarse procesos de actualización.
4. El *token* de identificación del cliente, que posibilita la asociación de solicitudes y respuestas.
5. Un *timestamp* que indica cuándo AWS IoT realizó la transmisión.

Todos estos campos estarán presentes en un mensaje transmitido por el servidor. La sintaxis de publicación de contenido es un poco más reducida, omitiéndose la *metadata* así como el *timestamp* de primer nivel. Para la interacción con un *Thing Shadow*, Amazon impone la utilización de un conjunto de tópicos reservados, en los que se intercambia información en el formato detallado anteriormente:

- /update
- /update/accepted
- /update/rejected
- /update/delta
- /get
- /get/accepted
- /get/rejected
- /delete
- /delete/accepted
- /delete/rejected

La ruta base para todos ellos es $aws/things/<nombre del dispositivo>/shadow y se puede evidenciar la utilización de un patrón: un tópico sin subniveles con un verbo que describe la operación, seguido de

un tópico con un subnivel de aceptación y otro de rechazo. El tópico sin subniveles es utilizado por los clientes para publicar mensajes con la finalidad de actualizar el estado, obtener la última versión disponible en el *broker*, o borrarlo. Por su parte, el servidor publicará en uno de los otros dos tópicos para reportar si el proceso fue exitoso o fallido. La operación de actualización es un caso especial debido a que es la única que requiere el envío de un objeto JSON (apegado a la sintaxis para *shadow devices*) y además tiene un tópico adicional, /update/delta. Ante una operación de actualización exitosa, el *broker* utilizará /update/delta para enviar la diferencia entre el estado deseado y el reportado.

Es importante recalcar que el formato de contenido descrito con anterioridad solo es necesario para utilizar el servicio de *thing shadows*. Al ser una implementación de MQTT, AWS IoT permite la definición de tópicos propios y el envío libre de datos en ellos.

- Motor de reglas.
  Provee procesamiento de mensajes e integración con otros productos de AWS, apoyándose en un lenguaje basado en SQL para seleccionar contenido de mensajes, procesar los datos, enviarlos a otros servicios o incluso republicarlos a otros suscriptores.

Un último aspecto importante de utilizar el servicio de Amazon es la imposición de seguridad. Cada dispositivo conectado requiere de una credencial, ya sea por medio de certificados X.509 o credenciales propias de AWS (temporales o permanentes), para poder acceder al *broker* de mensajes o al servicio de *Thing Shadows* (“AWS IoT Developer Guide”, s.f.). Gracias a esta autenticación de dispositivos, es posible aplicar políticas de seguridad que describen las acciones que una identidad específica puede llevar a cabo. Adicionalmente, todo el tráfico desde y hacia AWS IoT debe ser encriptado sobre TLS (*Transport Layer Security*).

2.2.4. XMPP

XMPP (*Extensible Messaging and Presence Protocol*) fue originalmente diseñado para aplicaciones de mensajería instantánea, basándose en el tremendo éxito y escalabilidad global de SMTP (*Simple Mail Transfer Protocol*), pero enfocándose en el envío de mensajes más pequeños con la menor latencia posible y sin persistencia (Waher, 2015, p. 126).

XMPP emplea lo que se denomina redes federadas de servidores XMPP, que actúan como *brokers* de mensajes para permitir la comunicación entre clientes detrás de firewalls. Cada servidor controla su propio dominio y autentica a los usuarios en él. Clientes pueden comunicarse con clientes en otros dominios gracias al uso de federaciones donde los servidores crean conexiones entre ellos, de una manera segura, para intercambiar mensajes entre sus dominios (Figura 10).

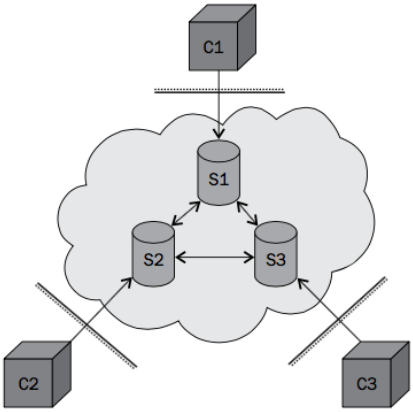


Figura 10. Red Federada XMPP
Tomada de Waher, 2015, p. 127.

Cuando un cliente se conecta, el servidor se asegura de que se autentique proveyendo sus credenciales correspondientes (de manera segura mediante el uso de SASL - *Simple Authentication and Security Layer*). La identidad de un cliente frecuentemente se denomina dirección XMPP o JID (Jabber ID). Cada conexión está también ligada a un recurso específico, que normalmente es

simplemente un cadena aleatoria de caracteres (cuyo propósito es permitir que un cliente pueda establecer varias conexiones simultaneas). La combinación del nombre de usuario, el dominio y el recurso constituyen el JID completo de la conexión (Figura 11). Este esquema de direccionamiento es una de las fortalezas del protocolo.

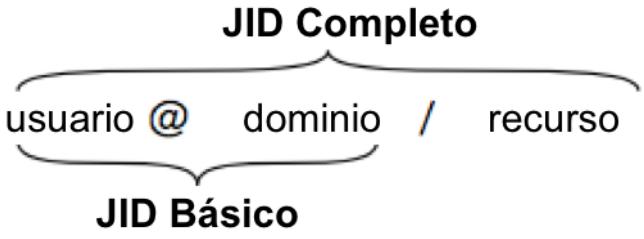


Figura 11. JID de una Conexión XMPP
Adaptada de Waher, 2015, p. 127.

XMPP no se limita a un único patrón de comunicación, lo cual se logra mediante sus tres primitivas de comunicación llamadas *stanzas*. La *stanza* de presencia es usada para enviar información propia a participantes interesados y autenticados. La segunda es la *stanza* de mensaje, destinada a enviar mensajes asíncronos a un receptor específico. La tercera es la *stanza* de IQ (*information*/*query*), utilizada para proveer el patrón de comunicación petición/respuesta. Apoyándose en estas primitivas un servidor XMPP puede proveer comunicación mediante publicación/suscripción, con el patrón *multicast* (que en XMPP lleva el nombre de chat multiusuario) o permitiendo que los clientes intercambien mensajes directamente punto a punto.

Otro aspecto importante es que los mensajes se intercambian en formato XML. El manejo de XML puede generar inconvenientes para clientes con limitada capacidad de procesamiento, por lo que es un punto que debe ser considerado al realizar una implementación.

Waher (2015, p. 125) manifiesta que, mientras MQTT se limita a un único patrón de comunicación y es útil para casos en los que la información publicada es

homogénea y tiene muchos potenciales consumidores, XMPP es apropiado para casos en los que se requiere de mensajes adaptados individualmente y en los que una comunicación bidireccional, en tiempo real, es importante.

2.3. Arduino Yún

Arduino es una plataforma *open-source* para la creación de prototipos, basada en hardware y software fácil de usar. El componente físico consiste de placas de circuito, programables mediante el envío de instrucciones a su microcontrolador. Para hacer esto se utiliza el lenguaje de programación Arduino, basado en un *framework* de programación para microcontroladores llamado Wiring, y su respectivo entorno de desarrollo (IDE) que lleva el nombre de Arduino Software. (“What is Arduino?”, s.f.)

Existen varios modelos de placas y el Yún, introducido en Mayo de 2013, es uno de ellos. Su característica distintiva es que posee dos procesadores: el primero es el Atmel ATmega32U4, ya presente en el Arduino Leonardo, un modelo clásico de la plataforma; y el segundo es el Atheros AR9331, en donde se ejecuta una distribución de Linux para sistemas embebidos llamada OpenWrt-Yun (basada en OpenWrt) (“Getting Started with the Arduino Yún”, s.f.). Adicionalmente, cuenta con un módulo WiFi integrado a la placa, a diferencia de otros modelos tradicionales que necesitan un componente externo para disponer de conectividad inalámbrica. En el lenguaje chino Yún significa nube, y es así que el objetivo de su creación fue el de facilitar la conexión directa a complejos servicios web (Romano, 2013).

El 32U4 presente en el Yun funciona de la misma manera que el del modelo Leonardo, con la excepción de que la conexión serial identificada como Serial1 no puede ser usada debido a que está reservada para la comunicación con el procesador AR9331. Para facilitar esta comunicación entre los dos procesadores se utiliza la librería Bridge (Figura 12).

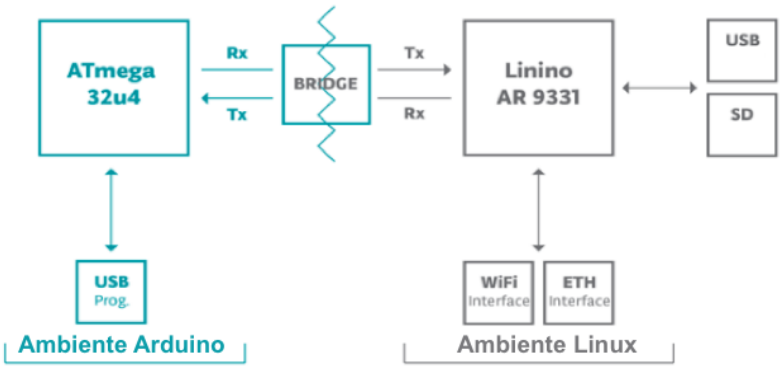


Figura 12. Relación entre procesador del Arduino Yún
Adaptada de “Getting Started with the Arduino Yún”, s.f.

2.4. Firmata

Firmata es un protocolo genérico para establecer comunicación con microcontroladores desde software en un ordenador host (Steiner, 2009). Su objetivo principal es convertir al hardware externo en una extensión del ambiente de programación presente en la computadora. Para esto se requiere que la plataforma externa ejecute un firmware Firmata y que la aplicación utilice una librería cliente que implemente el protocolo.

En una configuración tradicional la comunicación se logra mediante una conexión USB (*Universal Serial Bus*) en la que el microcontrolador aparece como un dispositivo serial para la aplicación host. Si bien es cierto que es extraño encontrar puertos RS232 en equipos modernos, USB puede presentarse como un RS232 o un equipo de puerto serial por lo que esta característica es aprovechada (Williams, 2014). La comunicación serial fue originalmente destinada a la transmisión de caracteres, motivo por el cual está orientada a bytes. En Firmata, el formato usado para comunicación de data son mensajes MIDI (*Musical Instrument Digital Interface*). Únicamente el formato de los mensajes, no el protocolo MIDI en su totalidad.

Aunque en teoría podría implementarse en cualquier plataforma, la versión para Arduino es su implementación más completa. Los origines de Firmata se dieron como un único firmware para Arduino que se convertiría en lo que hoy se conoce como Standard Firmata. Configurable Firmata es una variante que divide las diversas características soportadas por el protocolo en clases individuales, permitiendo mezclar funcionalidad personalizada con estándar y elegir únicamente las estrictamente necesarias (útil cuando la capacidad de almacenamiento es bastante limitada).

3. Desarrollo

3.1. Plan de Liberación

En la Tabla 1 se muestra el conjunto de historias de usuario que forman el *backlog* del producto, junto con su estimación respectiva y ciclo de desarrollo en el que se pretende inicialmente su consecución.

Tabla 1. Plan de Liberación

ID	**Historia**	**Iteración**	**Estimación (*Story Points* - Fibonnaci)**
H1	Como administrador deseo registrar nuevos dispositivos para que estén disponibles en el sistema.	1	1
H2	Como usuario final deseo visualizar en tiempo real el estado del calentador de agua para conocer si el dispositivo se encuentra encendido y cuál es la temperatura del agua.	1	3
H3	Como usuario final deseo crear eventos de encendido del dispositivo para un momento específico del día actual o siguiente.	1	3
H4	Como usuario final quiero especificar la temperatura que deberá ser alcanzada en los eventos y que el sistema encienda el calentador con la antelación apropiada para alcanzar dicho valor.	2	5
H5	Como usuario final quiero visualizar un listado de todos los eventos asignados al calentador de agua.	2	2

ID	Historia	Iteración	Estimación (*Story Points* - Fibonnaci)
H6	Como usuario final deseo que el acceso a mis dispositivos se encuentre restringido a un grupo seleccionado de usuarios.	3 – 4	2
H7	Como usuario final deseo obtener información histórica sobre el consumo eléctrico del calentador de agua para conocer la variación de dicho valor a través del tiempo.	3 – 4	5
H8	Como usuario final deseo definir la repetición de un evento previamente creado en base al día de la semana para no tener que crear eventos recurrentes a diario.	3 – 4	3
H9	Como usuario final quiero deshabilitar eventos temporalmente para evitar que sucedan sin la necesidad de eliminarlos.	3 – 4	1
H10	Como usuario final deseo editar la información de un evento en caso de haber introducido datos incorrectos o requerir la actualización de los mismos.	3 – 4	2
H11	Como usuario final quiero tener la posibilidad de eliminar eventos.	3 – 4	1
H12	Como usuario final deseo que eventos creados por mí no puedan ser editados, eliminados o deshabilitados por otros usuarios con acceso al calentador de agua.	3 – 4	2
Total			30

Para obtener este resultado se siguieron los pasos generales para la planificación de una liberación. De acuerdo a Mike Cohn, dichos pasos son los mostrados en la Figura 13.

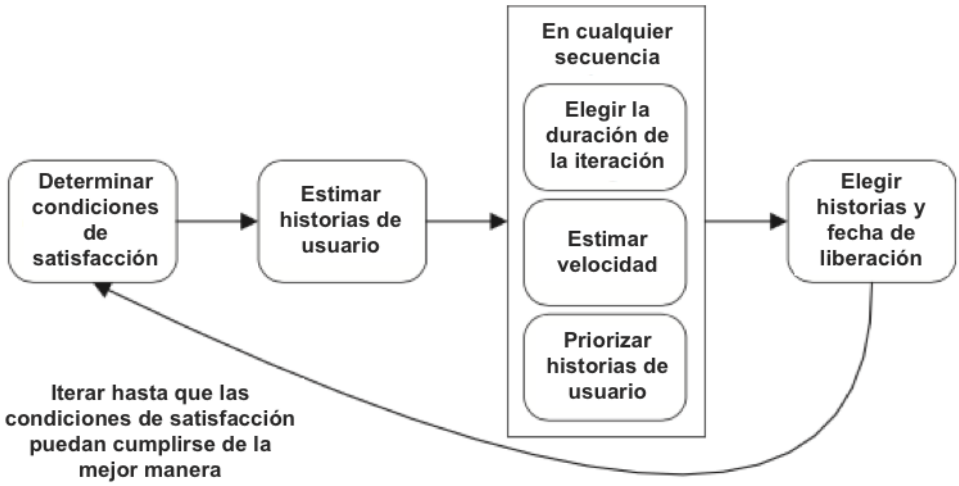


Figura 13. Pasos para Planificar un Release
Adaptada de Cohn, 2005, p. 133.

3.1.1. Condiciones de Satisfacción

Los objetivos del proyecto listados en la introducción del documento forman parte de las condiciones de satisfacción, es decir los criterios bajos los cuales el proyecto se evaluará como un éxito o un fracaso. Adicionalmente, hay que tomar en cuenta las restricciones que enfrenta el proyecto. Habiéndose establecido una duración inicial máxima de 2 meses y medio es evidente que se trata de un proyecto de fecha fija, por lo que otra de las condiciones de satisfacción es lograr los objetivos dentro del tiempo límite.

3.1.2. Estimación de Historias de Usuario

Para el segundo paso, se optó por puntos de historia como unidad de estimación, empleando como escala la secuencia de Fibonacci con un valor máximo de 8 y asignando valores relativos a partir de la identificación de la historia que se espera que represente el menor esfuerzo.

3.1.3. Elección de la Duración de las Iteraciones

La duración de cada iteración se definió en 2 semanas. Este valor se acopla bien a la duración del *release* (fijada por su fecha límite de entrega) y mantiene un sentido de urgencia junto con un costo aceptable asociado al proceso de iteración (algo conocido como *overhead of iterating*).

3.1.4. Estimación de Velocidad y Priorización de Historias

Cohn (2005, p. 175) también indica que existen tres opciones para estimar la velocidad de un equipo: usar valores históricos, útiles cuando un equipo tiene experiencia reciente trabajando juntos y las condiciones del proyecto previo fueron similares a las del actual; ejecutar una o más iteraciones con el objetivo de basarse en una velocidad observable; o realizar una predicción expandiendo un grupo representativo de historias en sus tareas constituyentes y planificando, a partir de ellas, un número determinado de *sprints* iniciales. En vista de no contar con información histórica y tampoco ser factible la elección de la segunda opción, se realizó una predicción planificando las dos primeras iteraciones mediante planificación impulsada por compromiso (cuyos pasos se pueden ver en la Figura 14). El método seleccionado obliga también a priorizar el *backlog* para establecer cuáles deberán ser las historias en las que se trabajará primero. En la Tabla 1 la prioridad de cada historia está dictada descendentemente por su orden de aparición, de más importante a menos importante.

Debido al grado de incertidumbre se decidió únicamente asignar historias específicas a los dos primeros ciclos de desarrollo. Indudablemente el plan de liberación podrá sufrir alteraciones por lo que invertir tiempo y esfuerzo en asignar trabajo a *sprints* específicos, posteriores a los dos primeros, no se consideró necesario.

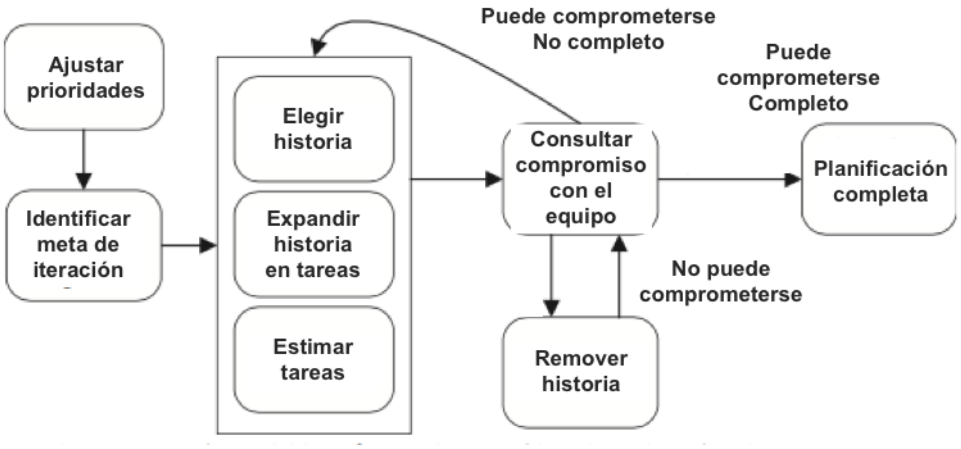


Figura 14. Actividades de Planificación de Iteración Impulsada por Compromiso (*Commitment-driven Iteration Planning*)
Adaptada de Cohn, 2005, p. 157.

Como se visualiza en la Tabla 1, los dos primeros *sprints* tienen 7 puntos de historia por lo que la velocidad promedio resultante es también 7.

3.1.5. Selección de Historias y Fecha de Liberación

Como ya se mencionó antes, la fecha límite se estableció como una restricción inicial. Utilizando los valores de duración de iteración y velocidad promedio estimada, se requerirían 5 ciclos de desarrollo para completar los 30 puntos que forman el *backlog* del proyecto. Si bien esto se encontraría dentro de los límites de tiempo, una de las iteraciones contaría con apenas dos puntos. Es así que se decidió ajustar la velocidad promedio esperada a 8 puntos de historia con el propósito inicial de finalizar la liberación en 4 *sprints*. Al poder cubrir la totalidad de historias, el proceso de selección de historias no fue necesario.

3.2. Ejecución

3.2.1. Resumen *Sprint 1* (Mayo 1 – Mayo 14)

3.2.1.1. Contenido

A continuación se detallan las historias planificadas para el primer *sprint*, los puntos asignados y los puntos logrados al concluir la iteración.

Tabla 2. Contenido Primer *Sprint*

ID de Historia	Resultado	Puntos Planeados	Puntos Ganados
H1	Pospuesta. Se asignó una prioridad inadecuada a la historia. No se cuenta con información suficiente sobre los servicios de AWS que serán usados por el producto para realizar efectivamente la historia.	1	0
H2	Iniciada; no finalizada.	3	0
H3	Iniciada; no finalizada.	3	0
Total		**7**	**0**

En la Figura 15 se muestra el gráfico de *burnup* del *sprint*. Si bien el alcance se redujo desde un inicio, el progreso fue nulo.

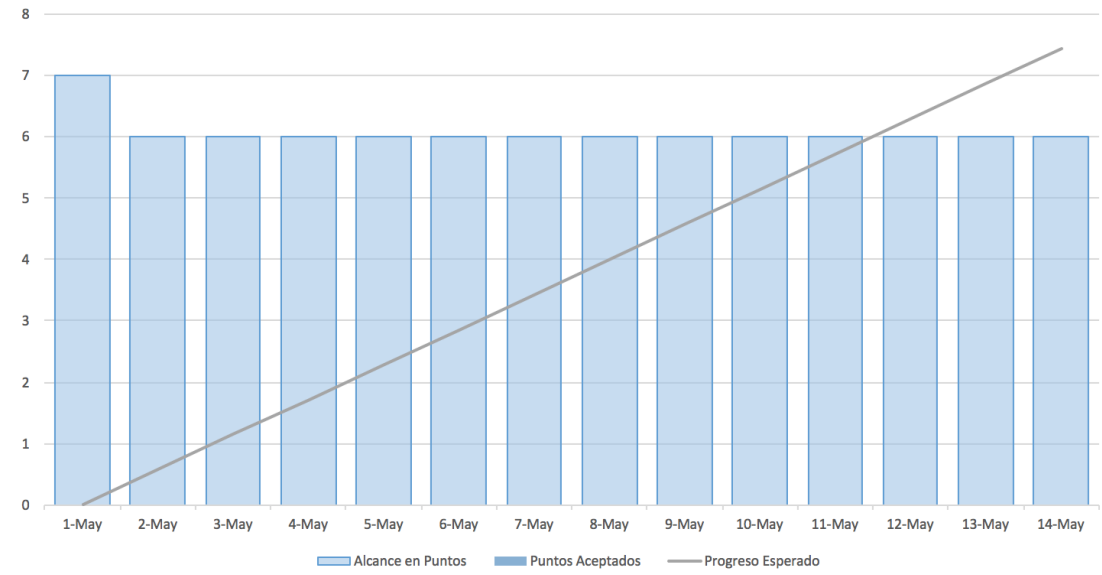


Figura 15. Burnup Primera Iteración

El alcance en puntos representa los puntos de las historias asignadas a la iteración, en un momento determinado en el tiempo. Los puntos aceptados corresponden a los de las historias finalizadas y verificadas como válidas. Y la recta de progreso esperado no es más que un avance ideal, obtenido de la división de la velocidad esperada (8 para este caso) por el número de días (14).

Por último, en la Figura 16 se puede visualizar el estado de las historias a través del tiempo. El rango de fechas mostrado en la gráfica excede la fecha de finalización del sprint (14 de Mayo) y permite observar que la aceptación de puntos iniciados en el primer *sprint* tomó lugar en el siguiente ciclo.

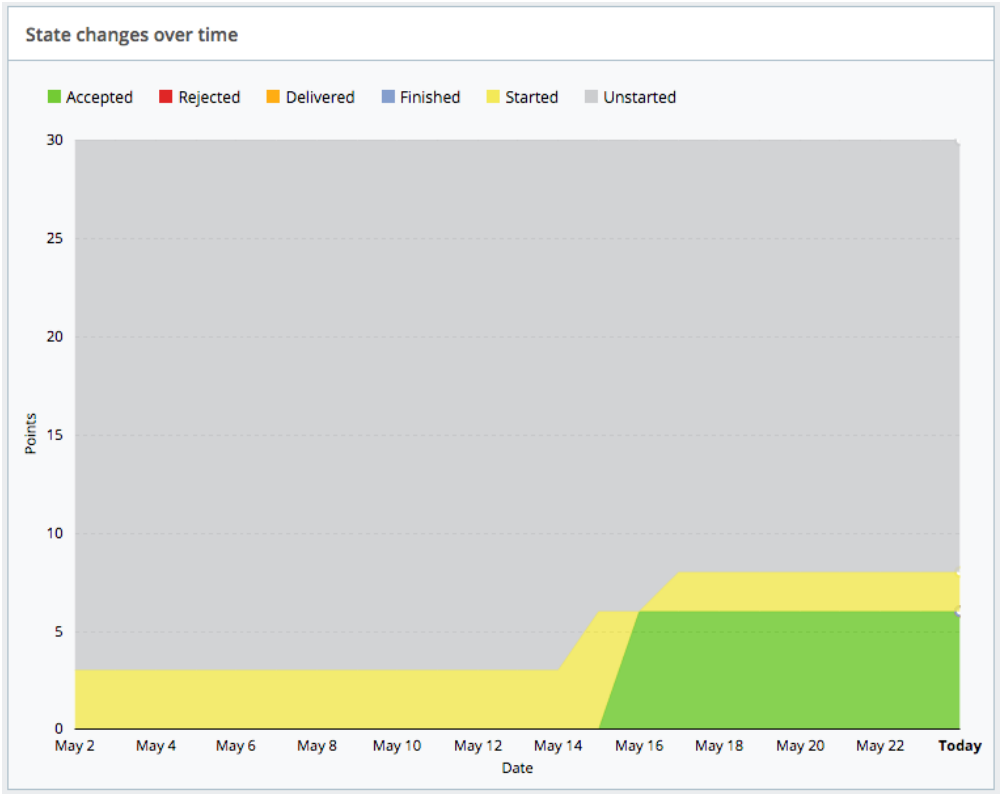


Figura 16. Variación de Estado de Historias (Mayo 1 – Mayo 24)

3.2.1.2. Retrospectiva

- ¿Qué se pudo hacer mejor?
  - Existieron dificultades para que el protocolo primario de comunicación (MQTT) funcione en la tecnología elegida para construir la interfaz de usuario. Se desperdició tiempo en esto y gran parte del esfuerzo se trasladó al final del *sprint*. Esto provocó que el progreso deseado no se alcance.
- Mejoras
  - Procurar no desperdiciar tiempo en detalles de implementación que no generan valor para el usuario.

No existieron aspectos positivos.

3.2.2. Resumen *Sprint* 2 (Mayo 15 – Mayo 28)

3.2.2.1. Contenido

A continuación se detallan las historias planificadas para el segundo *sprint*, los puntos asignados y los puntos logrados al concluir la iteración.

Tabla 3. Contenido Segundo *Sprint*

ID de Historia	Resultado	Puntos Planeados	Puntos Ganados
H2	Finalizada; iniciada en la anterior iteración.	3	3
H3	Finalizada; iniciada en la anterior iteración.	3	3
H5	Iniciada; no finalizada.	2	0
H6	Iniciada; no finalizada. Incluida en el *sprint* en vez de H4 luego de cambios en la prioridad.	2	0
Total		**10**	**6**

En la Figura 17 se muestra el gráfico de *burnup* del *sprint*. Se evidencia que 6 puntos se aceptaron al inicio del *sprint* y que este número de puntos aceptados se mantuvo constante hasta el 28 de Mayo. El progreso esperado sigue siendo el de la planificación inicial, 8 puntos, debido a que no existió velocidad observable en el sprint anterior.

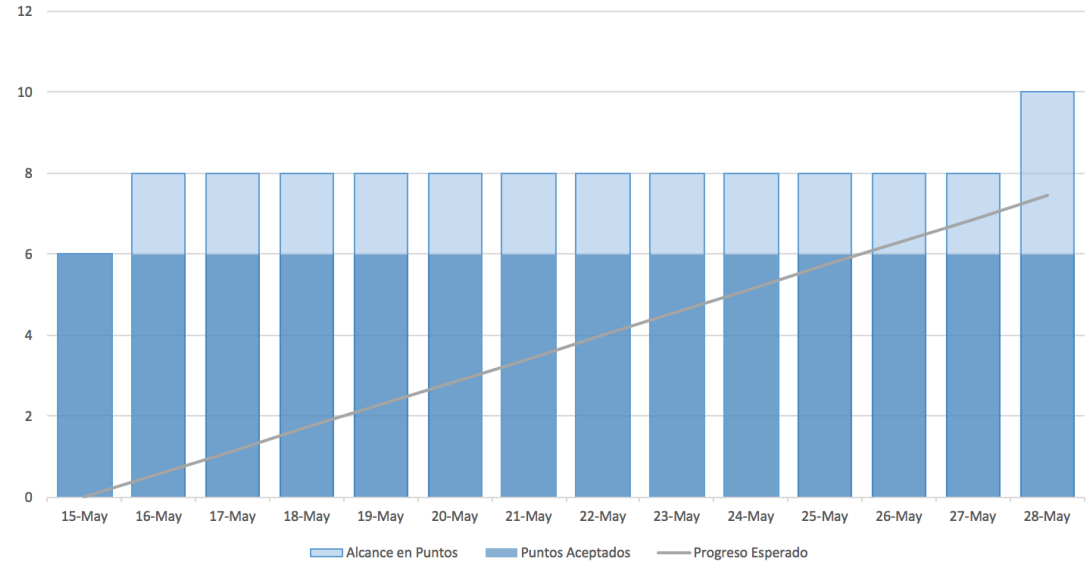


Figura 17. Burnup Segunda Iteración

3.2.2.2. Resultados

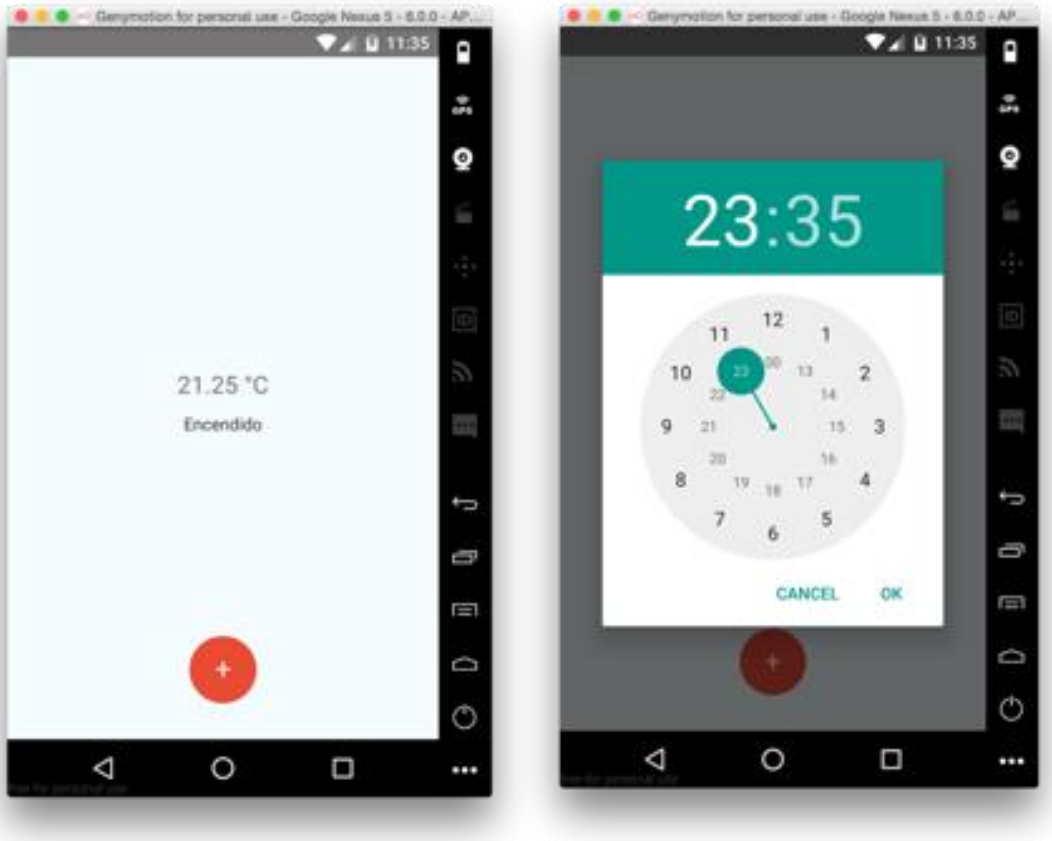


Figura 18. Resultados de Segunda Iteración

En la Figura 18 es posible observar capturas de la interfaz de usuario del aplicativo móvil correspondientes a las historias de usuario, H2 y H3, concluidas durante la iteración: la visualización de la temperatura y estado del tanque calentador de agua (a la izquierda de la captura); y la creación de eventos de encendido según una hora del día (a la derecha de la captura).

3.2.2.3. Retrospectiva

- ¿Qué se pudo hacer mejor?

    - Se evidenciaron los mismos inconvenientes que en el ciclo anterior: acumulación de esfuerzo al final de *sprint* y, como consecuencia, solo la finalización de las historias planificadas para el *sprint* 1.
- Mejoras
    - Procurar repartir el esfuerzo requerido en el *sprint* de manera más uniforme para que no se trasladen historias a futuros ciclos.

No existieron aspectos positivos.

3.2.3. Resumen *Sprint* 3 (Mayo 29 – Junio 11)

3.2.3.1. Contenido

A continuación se detallan las historias planificadas para el tercer *sprint*, los puntos asignados y los puntos logrados al concluir la iteración.

Tabla 4. Contenido Tercer *Sprint*

ID de Historia	Resultado	Puntos Planeados	Puntos Ganados
H5	Finalizada; iniciada en la anterior iteración.	2	2
H6	Finalizada; iniciada en la anterior iteración.	2	2
H4	Finalizada. Incluida en el *sprint* luego de cambios en la prioridad.	5	5
Total		**9**	**9**

En la Figura 19 se muestra el gráfico de *burnup* del *sprint*.

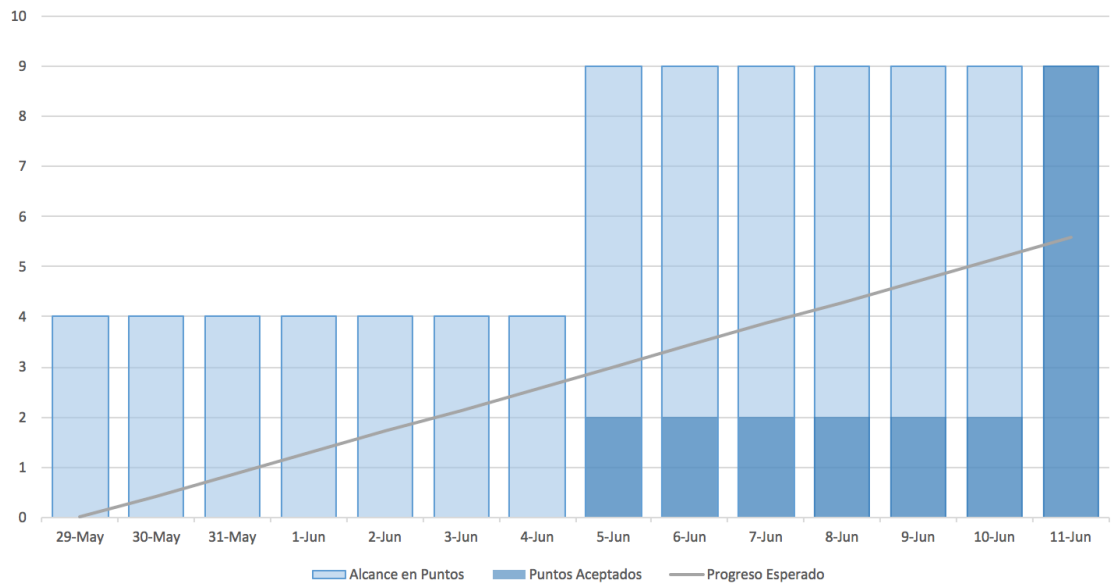


Figura 19. Burnup Tercera Iteración

A diferencia de la primera iteración en la que el progreso esperado venía dado por el valor de velocidad estimado durante la planificación del *sprint*, en la tercera iteración se emplea la velocidad observada en el ciclo anterior (6 puntos).

3.2.3.2. Resultados

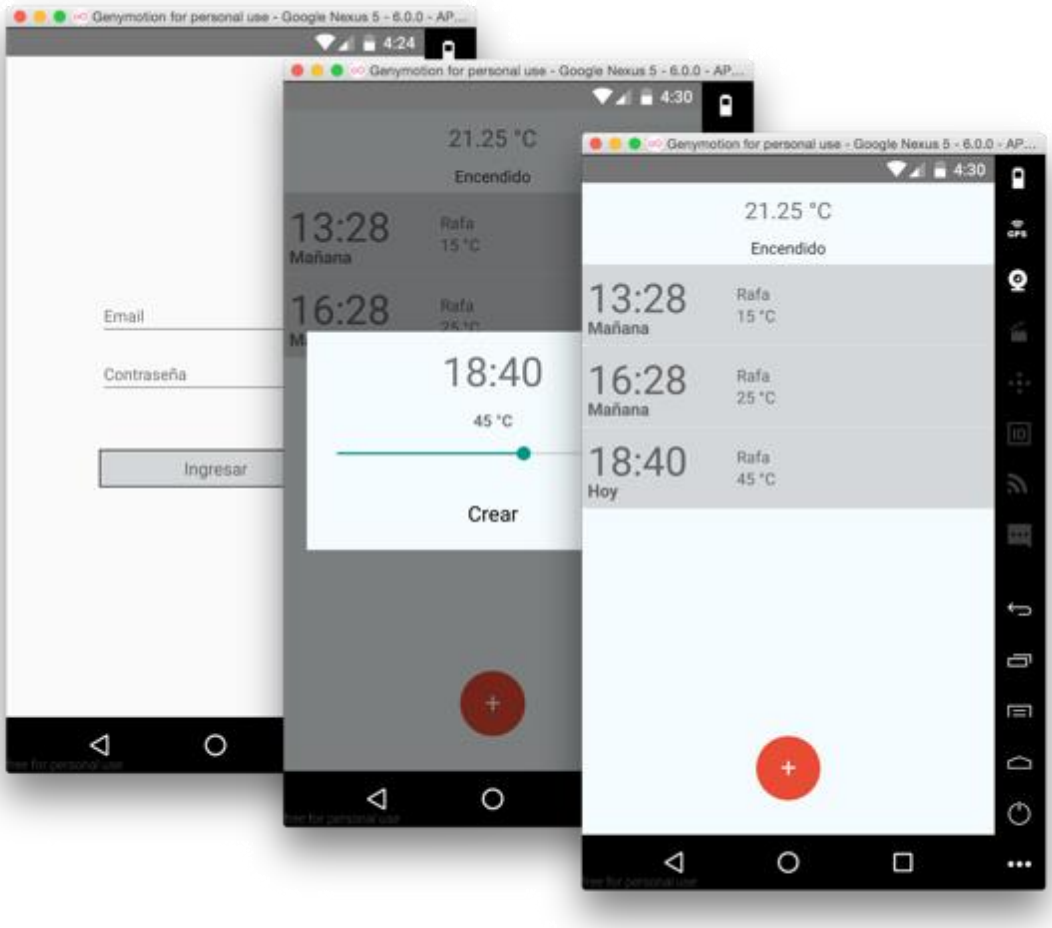


Figura 20. Resultados de Tercera Iteración

En la Figura 20 es posible observar capturas de la interfaz de usuario correspondientes a las historias de usuario H5, H6 y H4, concluidas durante la iteración.

3.2.3.3. Retrospectiva

- Positivo
    - Primera iteración con un avance real, ya que se completaron historias que no pertenecían al *sprint* anterior.

No se identificaron aspectos negativos, por lo que tampoco se plantean mejoras.

3.2.4. Resumen *Sprint* 4 (Junio 12 – Junio 25)

3.2.4.1. Contenido

A continuación se detallan las historias planificadas para el cuarto *sprint*, los puntos asignados y los puntos logrados al concluir la iteración.

Tabla 5. Contenido Cuarta Iteración

ID de Historia	Resultado	Puntos Planeados	Puntos Ganados
H7	Finalizada.	5	5
H8	Iniciada; no finalizada.	3	0
Total		**8**	**5**

En la Figura 21 se muestra el gráfico de *burnup* del *sprint*. Se evidencia que a lo largo de toda la iteración existió un alcance constante de 9 puntos de historia y superada la mitad del ciclo se aceptaron 5 puntos. El progreso esperado es de 9 puntos, correspondientes al número de puntos aceptados en la iteración previa.

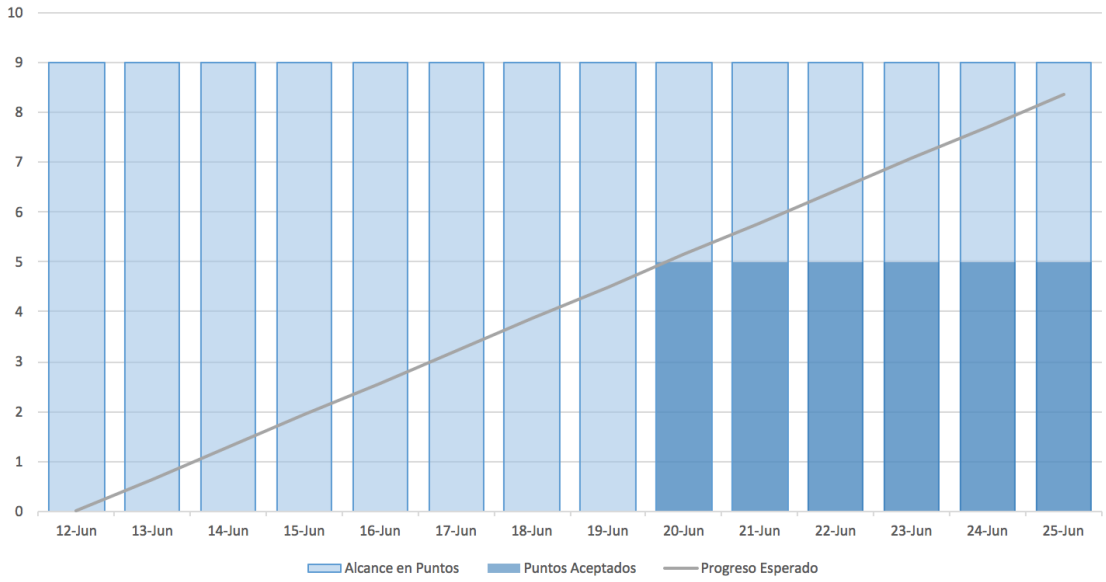


Figura 21. Burnup Cuarta Iteración

3.2.4.2. Resultados

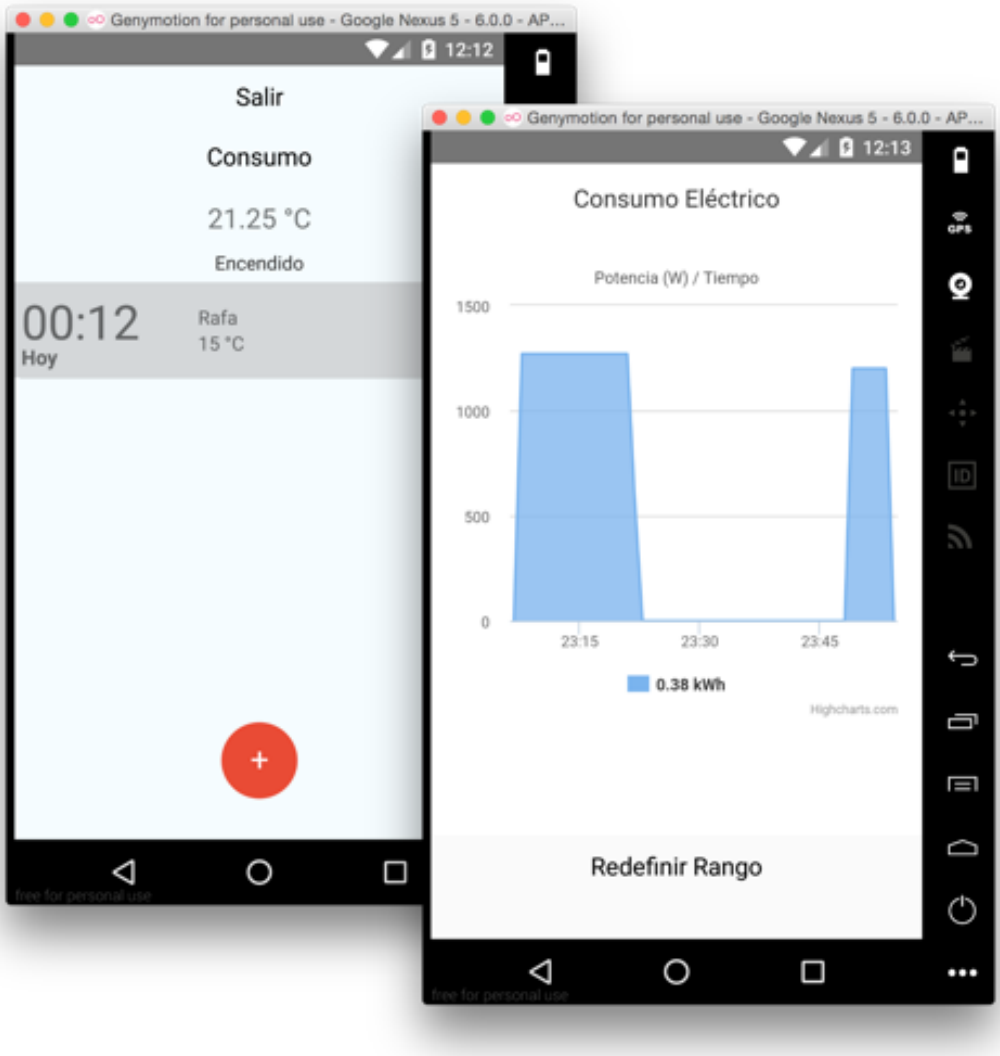


Figura 22. Resultado de Cuarta Iteración

En la Figura 22 es posible observar capturas de la interfaz de usuario correspondientes a la única historia de usuario concluida en la iteración: el despliegue de información histórica de consumo eléctrico.

3.2.4.3. Retrospectiva

- ¿Qué se pudo hacer mejor?
    - Como consecuencia del progreso nulo observado en el primer *sprint* no fue posible alcanzar la meta ideal de finalizar la liberación en solo 4 iteraciones. Por lo que no resultó adecuado redondear la velocidad estimada durante la planificación a 8 puntos.
- Mejoras
    - Respetar de mejor manera la velocidad calculada durante el proceso de planificación.

No se identificaron aspectos positivos.

3.2.5. Resumen *Sprint* 5 (Junio 26 – Julio 09)

3.2.5.1. Contenido

Contrario al plan de liberación inicial, fue necesario agregar un quinto ciclo de desarrollo para completar las historias de usuario planteadas. A continuación se detallan las historias planificadas, los puntos asignados y los puntos logrados al concluir la quinta iteración.

Tabla 6. Contenido Quinta Iteración

ID de Historia	Resultado	Puntos Planeados	Puntos Ganados
H8	Finalizada; iniciada en el *sprint* anterior.	3	3
H9	Finalizada.	1	1

H10	Finalizada.	2	2
H11	Finalizada.	1	1
H12	Finalizada.	2	2
H1	Finalizada. Incluida en el *sprint* debido a cambio en prioridades.	1	1
Total		**10**	**10**

En la Figura 23 se muestra el gráfico de *burnup* del *sprint*. A partir de la mitad del *sprint*, se evidencia un aumento de alcance y puntos aceptados.

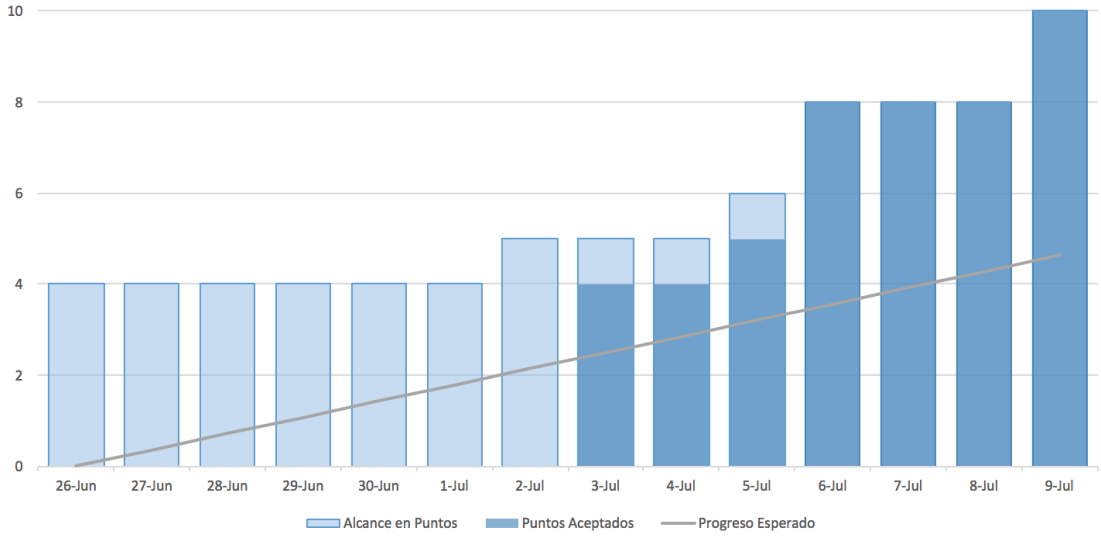


Figura 23. Burnup Quinta Iteración

3.2.5.2. Resultados

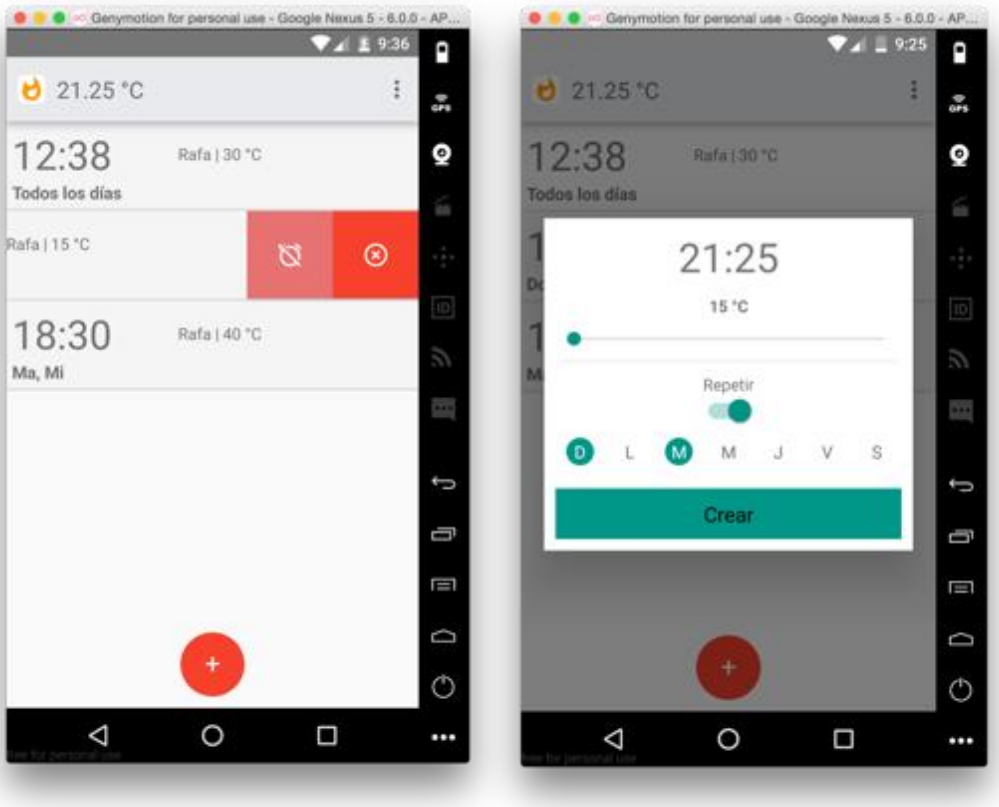


Figura 24. Resultado de Quinta Iteración

3.2.6. *Burndown* de la Liberación

El siguiente gráfico (Figura 25) representa las historias de usuario completadas a lo largo del transcurso del proyecto, en función de los puntos que se les asignaron.

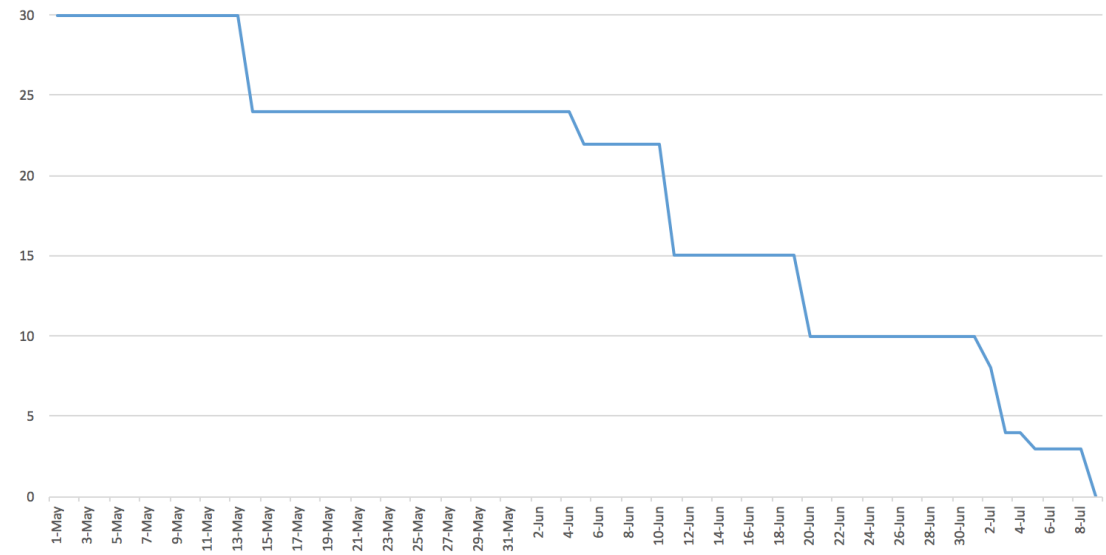


Figura 25. *Burndown* del *Release*

Como ya fue mencionado, no se logró el objetivo que completar la liberación en tan solo 4 *sprints*, que debieron concluir el 25 de Junio, como se planificó de forma inicial. En el gráfico es posible apreciar que las primeras iteraciones no presentaron un ritmo de progreso constante (se puede constatar lapsos de estancamiento), lo que repercutió negativamente en la velocidad de desarrollo y finalmente en la fecha de entrega.

3.3. Arquitectura

La arquitectura presentada a continuación es el resultado del proceso iterativo de desarrollo. Debido al nivel de incertidumbre inicial, no fue adecuado apegarse estrictamente a un esquema inicial. A medida que se realizaron las historias de usuario, se determinaron los elementos y mecanismos de interacción adecuados para obtener la funcionalidad deseada.

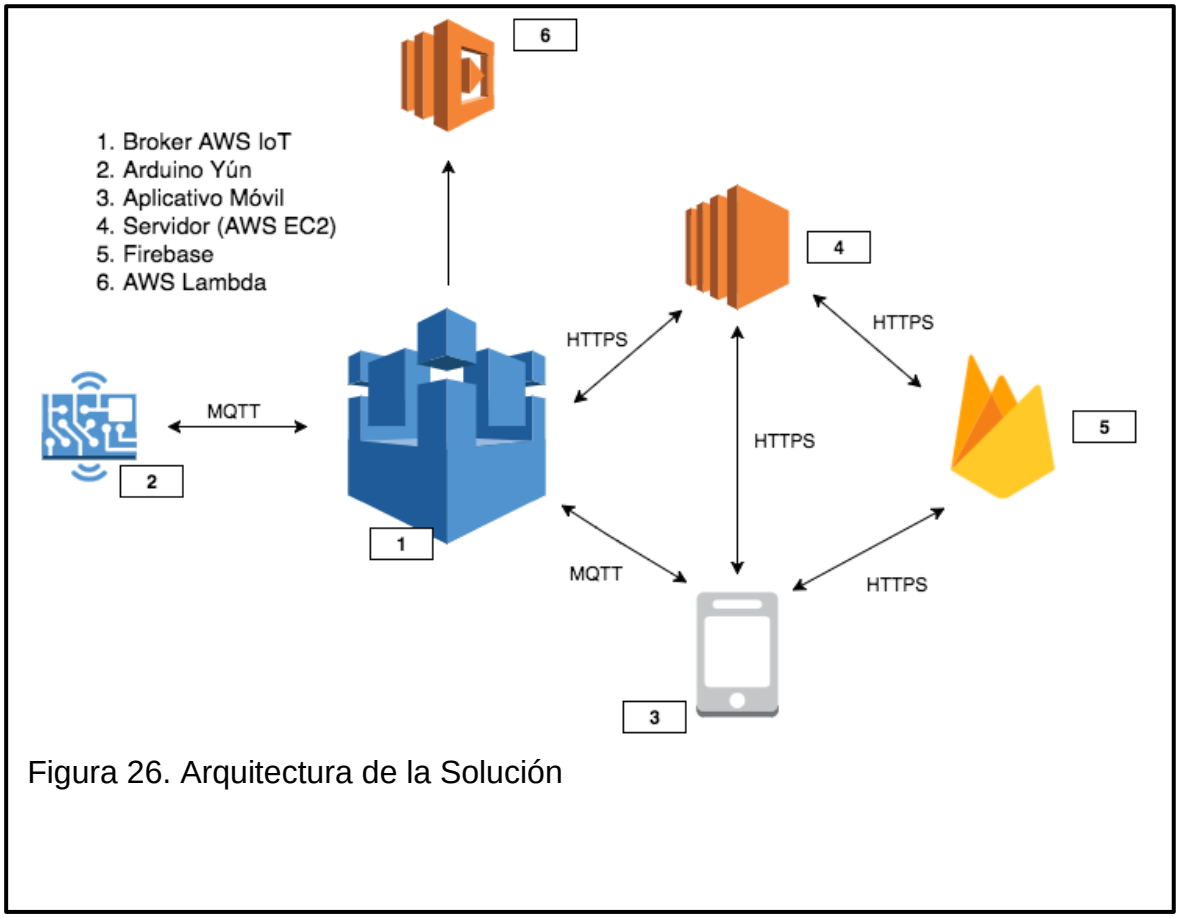


Figura 26. Arquitectura de la Solución

Cuatro componentes principales forman la solución (Figura 26): un aplicativo móvil destinado a ser la interfaz de usuario; un servidor que ejecuta procesos continuos para la calendarización de cambios de estado en el calentador, basándose en datos creados en el aplicativo; un *broker* MQTT; y un microcontrolador Arduino Yún que permite establecer la comunicación con el calentador de agua.

3.3.1. *Broker* MQTT

El *broker* MQTT, de AWS IoT, orquesta la distribución de mensajes enviados por los componentes restantes, que actúan como clientes. Adicionalmente, almacena el objeto JSON con la representación del estado del calentador de agua, como puede ser observado en la Figura 27. El estado del calentador está compuesto por dos campos sencillos: *enabled*, un booleano para indicar si está encendido o apagado; y *waterTemperature*, que contiene el valor numérico de la temperatura del agua contenida en el tanque (en grados centígrados).

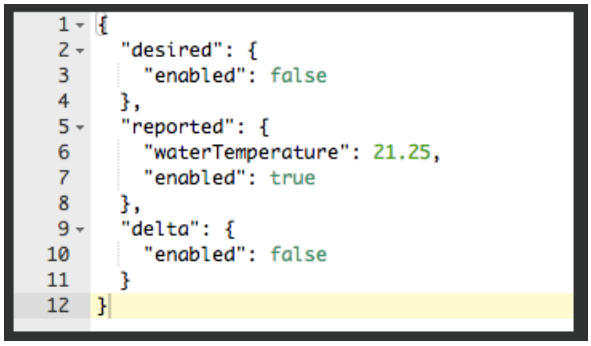


Figura 27. Estado del Calentador de Agua

3.3.2. Arduino

Como se mencionó en el marco teórico, una configuración tradicional para trabajar con Firmata implica una conexión USB que será reconocida como serial. Habitualmente esto implica tener al microcontrolador conectado, por USB, a un ordenador. Sin embargo, es posible aprovechar el procesador Atheros del Arduino Yún, y su conexión serial con el 32U4, con el fin de que desempeñe el rol de la computadora y así brindar independencia a la placa. De esta manera, el Atheros puede recibir información de un sensor de temperatura DS18B20 y un sensor de corriente ACS712, así como también controlar un módulo relé conectado a uno de los pines digitales del Arduino (Figuras 28 y 29). El software utilizado para conseguir esta interacción es:

- Firmware *Configurable Firmata* en el lado del procesador Atmel. *Configurable Firmata* es necesario debido a que el sensor de temperatura se trata de un dispositivo que utiliza un bus de comunicación 1-Wire que no tiene soporte en la versión estándar de Firmata.

- Una librería cliente Firmata implementada en JavaScript. Dicha librería es utilizada por una aplicación que emplea Node JS como su entorno de ejecución, en el lado Linux del Arduino (en el procesador Atheros). La librería elegida lleva el nombre de Cylon JS.

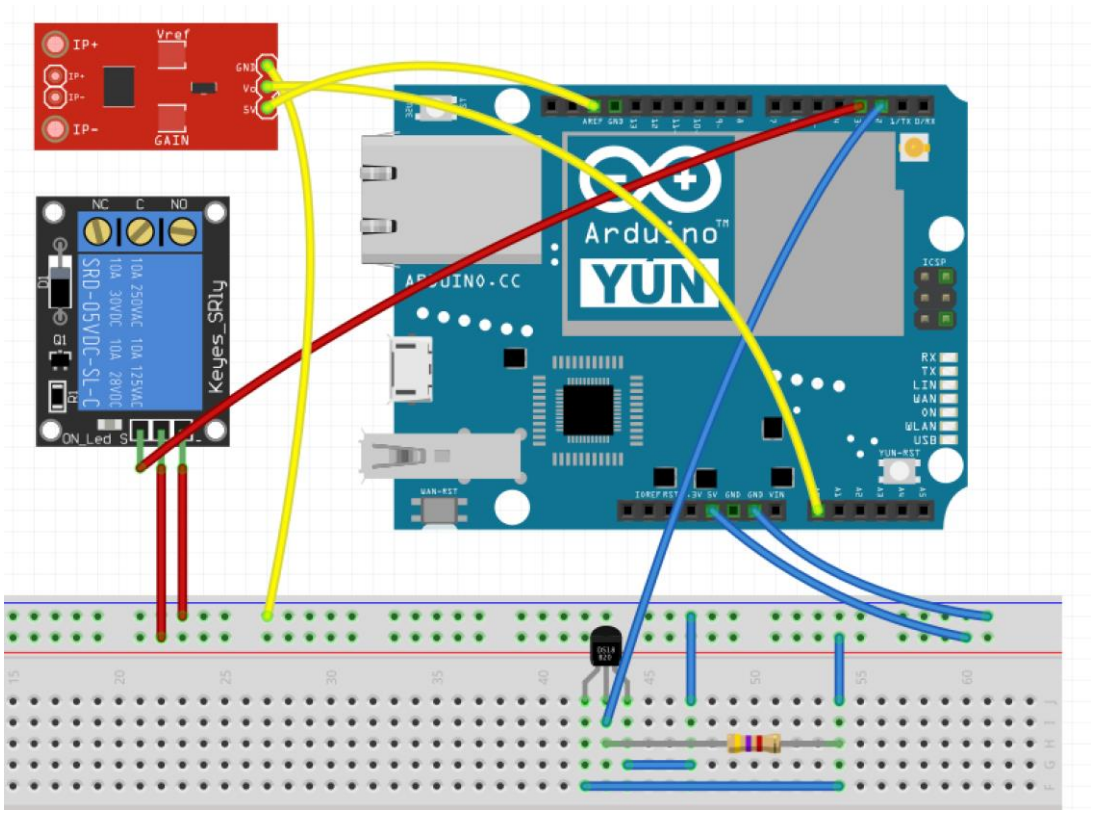


Figura 28. Circuito Arduino sin Carga

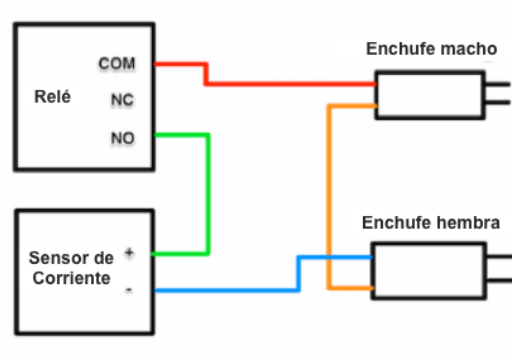


Figura 29. Conexión entre ACS712 y Relé
Adaptada de Schwartz, 2014.

El otro componente de software utilizado en la aplicación JavaScript, ejecutada en el Arduino, es el módulo cliente de AWS IoT para JavaScript llamado aws-iot-device-sdk. La utilización de esta librería otorga la posibilidad de iniciar un cliente MQTT y utilizar el servicio de *Thing Shadows* desde el Yún. Para que la comunicación pueda realizarse exitosamente es necesario que el Arduino sea un dispositivo único registrado en AWS IoT y cuente con un certificado X.509, además de una política atada a él, para identificar los mensajes provenientes del microcontrolador y realizar las autorizaciones respectivas. El certificado X.509, su llave privada y el certificado CA raíz de Amazon se almacenan en un directorio del dispositivo (Figura 30), y son parámetros necesarios para la inicialización del cliente MQTT. El certificado CA se incluye para que el Arduino no se conecte a servidores impostores que pudieran querer hacerse pasar por el de AWS IoT.

```
root@Arduino:/mnt/sda1/arduino/www# ls -l certificates/
-rwxr-xr-x    1 root     root          1220 Apr 17 20:19 b954e6c1e1-certificate.pem
-rwxr-xr-x    1 root     root          1675 Apr 17 20:19 b954e6c1e1-private.pem
-rwxr-xr-x    1 root     root          1758 Apr 17 20:19 rootCA.pem
```

Figura 30. Certificados Arduino

En la consola de administración de AWS IoT es posible observar qué “cosa” y política de seguridad están vinculadas al certificado. En la Figura 31 se observa que el certificado de la Figura 30 está vinculado a la política *PubSubToAnyTopic*,

que otorga acceso total a todas las acciones disponibles en el servicio de IoT, e identifica a *water-heater*.

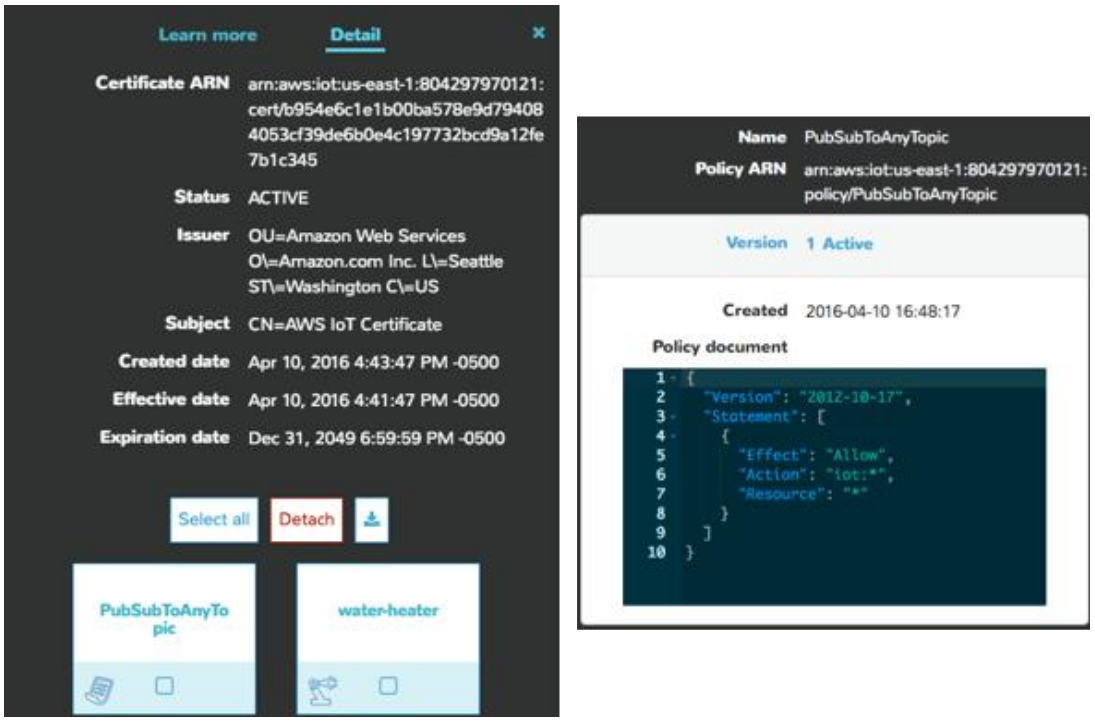


Figura 31. Política y Dispositivo Ligado a Certificado

La publicación de información e interacción con el servidor de mensajes es descrita en las siguientes secciones.

3.3.2.1. Temperatura

Valores discretos son leídos en un intervalo constante de veinte segundos y publicados al tópico *update* del dispositivo, en el atributo de *waterTemperature* del estado reportado (Figura 32).

```
clientTokens.update = thingShadows.update(WATER_HEATER_THING_NAME, {
  state: {
    reported: {
      waterTemperature: temperature
    }
  }
});
```

Figura 32. Actualización de Temperatura

3.3.2.2. Potencia Efectiva

Dado que la corriente es una señal análoga es necesario realizar un muestreo para posteriormente calcular la raíz cuadrática media de su valor efectivo, que a su vez puede multiplicarse por el voltaje del suministro eléctrico para obtener la potencia efectiva consumida.

La raíz cuadrática media, RMS por sus siglas en inglés, de una onda sinusoidal provoca el mismo efecto de calentamiento que una corriente directa, DC, del mismo valor (“RMS Voltage Tutorial”, s.f.). En otras palabras, es un valor DC que indica a cuantos voltios o amperes directos equivale una onda sinusoidal que varía en el tiempo, en términos de su capacidad de producir la misma potencia. Por este motivo, al resultado se lo conoce como valor efectivo. De acuerdo al método de resolución analítico, la corriente efectiva (I_{RMS}) se obtiene aplicando la Ecuación 1, el producto del pico de la onda (I_m) por uno dividido para la raíz cuadrada de dos.

$$I_{RMS} = I_m (1 / 2^{1/2}) \quad \text{(Ecuación 1)}$$

El pico de la onda se puede obtener tomando muestras de valores instantáneos. Dado que el voltaje es un valor conocido, de acuerdo a lo que el suministro eléctrico provea al calentador de agua (120 voltios para este caso), es sencillo obtener la potencia efectiva multiplicando la corriente efectiva por el voltaje (Ecuación 2).

$P = I_{RMS} \times 120$ (Ecuación 2)

La potencia efectiva es publicada, a un tópico no reservado con el mismo intervalo que la temperatura, junto con un *timestamp*, un identificador del dispositivo y un valor booleano que indica si es una medición inicial (Figura 33); sin embargo, este proceso solo se lleva a cabo cuando el relé se encuentra cerrado (lo que para el caso del circuito del proyecto significa que el calentador está encendido).

```
my.getEffectivePower()
  .then(function (effectivePower) {
    var message = {
      power: effectivePower,
      timestamp: Date.now(),
      deviceId: WATER_HEATER_THING_NAME,
      initialMeasurement: initialMeasurement
    };

    initialMeasurement = false;
    thingShadows.publish('measurements/power', JSON.stringify(message));
  })
```

Figura 33. Publicación de Potencia Efectiva

Una regla en el *broker* será activada cuando un mensaje con destino a ese tópico sea recibido, procesará la información usando una función Lambda de AWS y finalmente almacenará los datos en la tabla *PowerMeasurements* en DynamoDB (Figura 34).

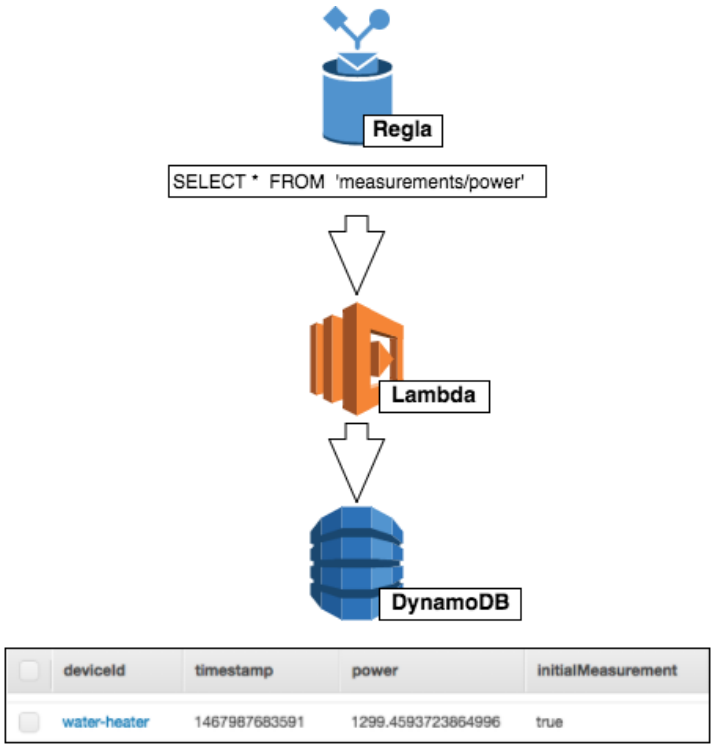


Figura 34. Procesamiento de Información de Potencia Efectiva

DynamoDB es una base de datos NoSQL por lo que cada uno de los ítems almacenados no necesariamente debe apegarse a un esquema estricto. No obstante, como en otras bases de datos, se requiere de una clave primaria para identificar únicamente a los elementos almacenados. En el caso de *PowerMeasurements*, se tiene una clave compuesta por:

- *Partition Key – deviceId*: Este valor se usa en una función interna de *hash* que determinará la partición en la que el ítem se guardará.
- *Sort Key – timestamp*: Todos los elementos con la misma clave de partición se almacenan juntos, ordenados por su respectivo valor de clave de ordenamiento.

La Figura 35 presenta un ítem almacenado en la tabla, junto con sus tipos de dato.

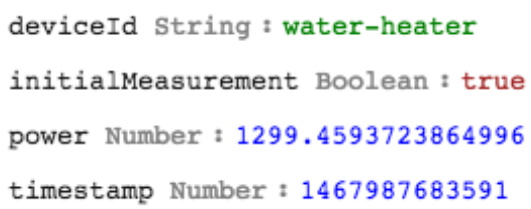


Figura 35. Ítem de *PowerMeasurements*.

3.3.2.3. Encendido / Apagado

El microcontrolador también puede responder a cambios en el estado del dispositivo por medio de la evaluación del contenido publicado al tópico *update/delta*, al que está suscrito. Al momento de producirse una diferencia entre el valor reportado y requerido para el atributo *enabled*, el broker publicará un mensaje al tópico antes mencionado y, dependiendo de si es verdadero o falso, el Arduino encenderá o apagará el calentador de agua (respectivamente). Una vez concluido el encendido o apagado se procede a actualizar el estado reportado, sincronizándolo con el valor deseado.

Un proceso adicional que se lleva a cabo es la publicación del estado actual (tanto *enabled* como *waterTemperature*) bajo el atributo *reported* cuando el dispositivo establece exitosamente una conexión con el *broker*. Esto puede darse cuando el Arduino es encendido o ante una reconexión y, debido a la capacidad de reacción ante cambios en el estado, permite que el dispositivo se sincronice con el estado almacenado en la nube (*Thing Shadow*).

3.3.3. Aplicación Móvil

La aplicación móvil, desarrollada en React Native, representa la interfaz para el usuario final. Se apoya en Firebase, un *backend as a service*, para autenticación y persistencia.

Una vez que el usuario se ha autenticado exitosamente contra Firebase, un proceso adicional se inicia con la finalidad de obtener credenciales temporales que permitan a la aplicación conectarse directamente al servidor MQTT (Figura 36). El servicio de Amazon con el que hay que interactuar para lograr esto se denomina Cognito. Cognito recibirá como parámetro el identificador único otorgado por Firebase con el fin de generar una identidad válida para Amazon. Una vez que esta identidad se ha generado, se podrá acceder a credenciales temporales ligadas a ella, que podrán ser utilizadas para el consumo directo de servicios de AWS (de acuerdo a los permisos atados a la identidad). De esta manera, la aplicación podrá suscribirse a los tópicos pertenecientes al dispositivo al que el usuario tiene acceso. Esta suscripción permitirá el despliegue, en tiempo real, de información sobre la temperatura y estado habilitado o deshabilitado del calentador de agua.

Como se describió en el caso del Arduino, también es posible autenticarse ante AWS IoT mediante la utilización de certificados X.509; sin embargo, no es un método adecuado para este escenario debido a que implicaría generar y almacenar certificados únicos para cada uno de los aplicativos cliente que pretendan conectarse al *broker* MQTT. Además, la validez temporal de las credenciales representa una medida adicional de seguridad. Lastimosamente, el proceso no está pensado para ser realizado solo del lado del cliente móvil debido a que requiere utilizar credenciales administrativas de Amazon para la generación de las identidades temporales. Esto obliga a mover dicha funcionalidad a un servidor adicional y añade complejidad a la solución.

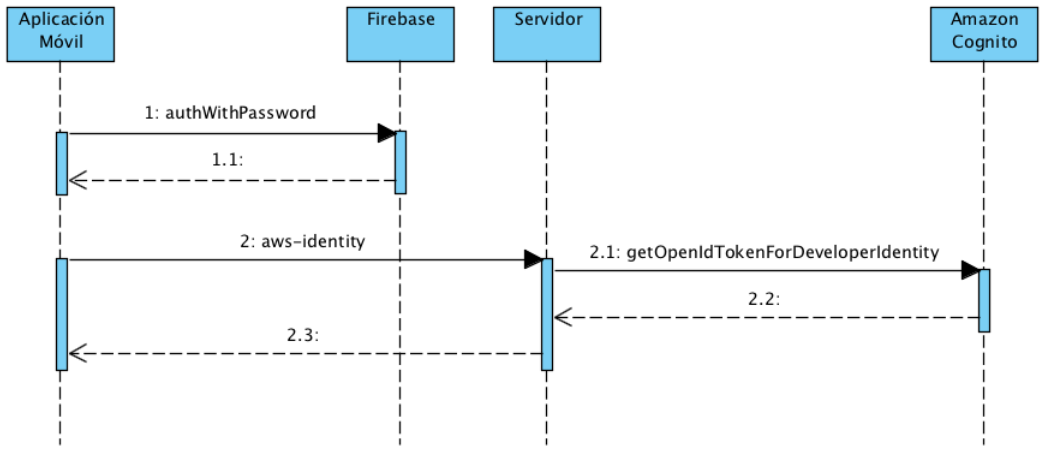


Figura 36. Autenticación Aplicación Móvil

Una de las características principales de Firebase es su base de datos NoSQL alojada en la nube y que permite almacenar y sincronizar datos en tiempo real, enviando actualizaciones automáticas de cambios en los datos a todos los clientes conectados. Debido a lo antes descrito, se la consideró adecuada para poder ofrecer al usuario una experiencia de manejo de información en tiempo real para los eventos de encendido del calentador de agua. Uno de estos eventos es sencillamente una hora del día en la que el usuario desea que el agua contenida en el tanque alcance cierta temperatura.

En Firebase, los datos son guardados simplemente como un árbol JSON. Para los eventos de encendido se tiene la estructura de la Figura 37.

En el atributo *profiles* se almacena el nombre de los usuarios, teniendo un identificador autogenerado por Firebase como llave de los registros. La información básica requerida del usuario (correo y contraseña) es administrada directamente por Firebase y no es desplegada en las consultas. Por su parte, el atributo *events* contiene los datos de los eventos, relacionados a un usuario mediante el campo *user* en el que se almacena su identificador único.

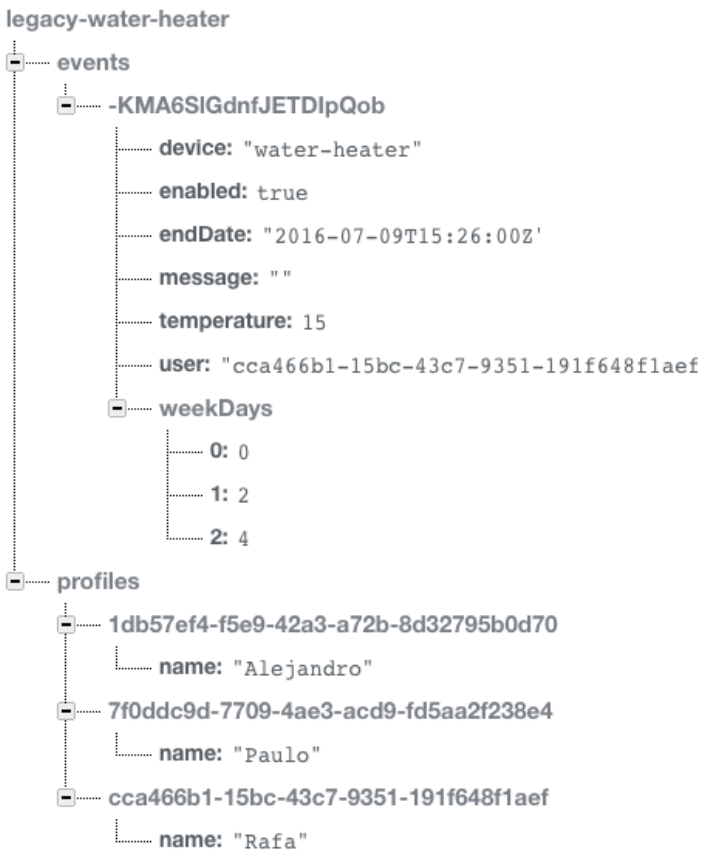


Figura 37. Estructura Firebase para Eventos de Encendido

Por último, el usuario puede acceder a otra funcionalidad que no requiere de interacción directa con el *broker* de mensajes: consultas de consumo eléctrico, basadas en un rango de fechas.

3.3.4. Servidor

El último componente, el servidor, se apoya en Firebase para reaccionar ante la creación, modificación o eliminación de eventos de encendido. Para brindar esta funcionalidad, uno de sus procesos es un cliente Firebase.

Con la información del evento se calcula un estimado de cuándo deberá encenderse el calentador para alcanzar la temperatura solicitada. Esta estimación se realiza mediante un modelo matemático en el que la razón de cambio de temperatura, para el sistema de la Figura 38, es igual a la razón de calor producido por el elemento generador de calor menos la razón de pérdida de calor ante el medio ambiente. Se asume las siguientes condiciones:

- Existe circulación adecuada del líquido dentro del tanque de manera que la temperatura se distribuye uniformemente.

- La pérdida de calor en la superficie está dada por la ley de enfriamiento de Newton y la temperatura del ambiente es constante.

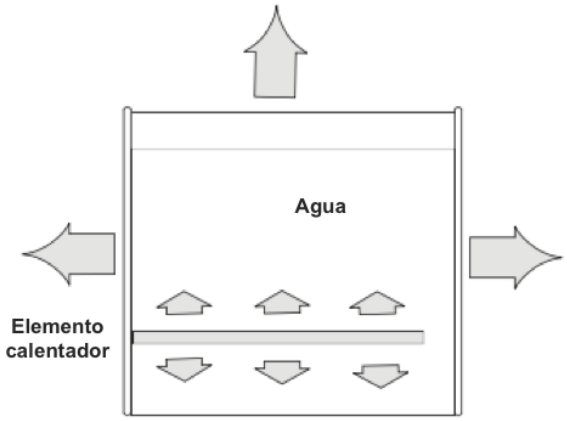


Figura 38. Diagrama de un Calentador de Agua
Adaptada de Barnes y Fulford, 2008, p. 222.

Dicho modelo, acorde a lo demostrado por Barnes y Fulford (2008, p. 224), se traduce a la Ecuación 3.

$$cm(dU/dt) = q - hS(U(t) - u_s) \quad \text{(Ecuación 3)}$$

Donde c es el calor específico del agua, m es la masa del agua, U(t) denota la temperatura en un tiempo t, q es la razón constante de producción de calor del elemento generador de temperatura, h es el coeficiente de enfriamiento de Newton, S es el área de superficie del calentador y u_s es la temperatura constante del ambiente.

Otra suposición añadida al modelo es que la pérdida de temperatura es despreciable. Esto puede ser logrado aislando el tanque con un material como fibra de vidrio. Como resultado, la Ecuación 3 se simplifica a la Ecuación 4 (Donde u_0 es la temperatura inicial del agua).

$$U(t) = qt/cm + u_0 \quad \text{(Ecuación 4)}$$

Para el caso del proyecto, se definió la temperatura inicial como una constante de 15 grados centígrados, correspondientes a la mínima temperatura esperada en el tanque. Así, sabiendo la temperatura final que se desea conseguir es posible despejar el tiempo necesario para llegar a tal valor.

Resulta evidente que este tiempo es meramente una aproximación. Modelar con gran precisión la temperatura del líquido contenido en el tanque no es un proceso trivial y excede el alcance del proyecto.

Luego de completada la estimación, dos tareas programadas serán creadas: una para encender el dispositivo (con la anticipación calculada) y otra para apagarlo en la hora indicada en el evento. Esta calendarización se podría realizar utilizando un cron; sin embargo, para desacoplar esta funcionalidad del sistema operativo se eligió una librería para Node JS, llamada Agenda, que persiste las tareas a una base de datos MongoDB. Un proceso adicional, de Agenda, corre

continuamente con el propósito de ejecutar las tareas programadas. Esta ejecución consiste en realizar una llamada al *endpoint* REST del tópico *update* con el estado correspondiente (habilitado o deshabilitado). No se utiliza un cliente MQTT para estos cambios de estado debido a que son interacciones puntuales, para las cuales un patrón petición/respuesta es apropiado.

Por último, el servidor también es el encargado de exponer información del consumo eléctrico del tanque. Tal como se indicó en la sección 2.4.2.2, en una tabla de DynamoDB se almacena información de potencia efectiva. No obstante, esa información debe ser procesada antes de tener algún significado para el usuario. Lo que se necesita es la integral de la potencia en función del tiempo. No se tiene la función exacta que describe tal relación, pero se cuenta con valores puntuales de potencia a través del tiempo por lo que es posible utilizar la regla del trapecio para estimar el área bajo la curva. La unidad del valor de energía resultante es el joule. Para transformarlo a kilovatio por hora basta con dividirlo para 3 600 000.

En resumen, utilizando Node JS como plataforma, el servidor ejecuta tres procesos distintos:

- Un servidor HTTP que ofrece datos relacionados a AWS y al consumo de energía eléctrica.
- Un proceso continuo Agenda, para tareas programadas relacionadas a los eventos de encendido del calentador de agua.
- Un proceso continuo con un cliente Firebase, para recibir notificaciones de acciones sobre la base de datos en tiempo real y actuar correspondientemente.

3.4. API

Por definición, un API (*Application Programming Interface*) provee una abstracción para un problema y especifica cómo clientes deben interactuar con los componentes de software que implementan la solución a aquel problema. En

esencia, APIs definen bloques constitutivos reutilizables que permiten incorporar piezas modulares de funcionalidad a aplicaciones de usuario final. (Reddy, 2011, p. 1)

Para consultar el consumo eléctrico se expone un *endpoint* HTTP, documentado (utilizando el formato API Blueprint) en la Figura 39 y 40.

Las interacciones para consultar y manipular el estado del calentador de agua se realizan primariamente sobre el protocolo MQTT. Para conocer el estado en un momento puntual del tiempo, se debe enviar un mensaje al tópico $aws/things/<nombre del dispositivo>/shadow/get y estar suscripto al tópico $aws/things/<nombre del dispositivo>/shadow/get/accepted para recibir la respuesta. Si se desea obtener actualizaciones en tiempo real, sobre el estado del tanque, es necesario suscribirse al tópico $aws/things/<nombre del dispositivo>/shadow/update/accepted. Finalmente, si se busca actualizar el estado se debe publicar mensajes al tópico $aws/things/<nombre del dispositivo>/shadow/update, siguiendo el formato indicado en la sección 2.4.1.

Figura 39. Ejemplo de Respuesta *Endpoint* Consumo Eléctrico

```
# Calentador de Agua

## Consumo Eléctrico [/energy-consumption?timezone, startDate, endDate, deviceId, format]

### Lista de medidas y energía total [GET]

+ Parameters

    + timezone (string, optional) - Identificador válido de zona horaria.

        + Default: `America/Bogota`

    + startDate: `2016-06-18` (string, required) - Fecha inicial de búsqueda.

    + endDate: `2016-06-19` (string, required) - Fecha final de búsqueda.

    + deviceId: `water-heater` (string, required) - Nombre de disposivito registrado en AWS IoT.

    + format (enum[string], optional)

        + Default: `json`

        + Members
            + `json`
            + `html`
```

Figura 40. Documentación *Endpoint* Consumo Eléctrico

3.5. Pruebas de Aceptación

En la Tabla 7 se muestran las pruebas de aceptación del sistema. En el Anexo 1 se presenta la secuencia de pantallas correspondientes a cada prueba.

Tabla 7. Pruebas de Aceptación.

ID	Descripción	Resultados Esperados	Resultados Obtenidos
P1	Precondición: Iniciar sesión. - Partiendo del calentador de agua en estado apagado forzar manualmente su encendido.	Antes de forzar el encendido el aplicativo móvil deberá mostrar el estado del calentador como apagado. Una vez encendido, el estado en la interfaz gráfica deberá cambiar a encendido. La temperatura desplegada deberá	Aprobada.

ID	Descripción	Resultados Esperados	Resultados Obtenidos
		aumentar conforme avanza el tiempo.	
P2	Precondición: Iniciar sesión. - Seleccionar el componente gráfico destinado a agregar nuevos eventos. - Seleccionar una hora en el componente de reloj y aceptar dicha selección.	Un evento, con la hora seleccionada, deberá ser creado y desplegado en la interfaz gráfica. - Si la hora elegida es igual o menor a la hora actual, un evento para el día de mañana deberá ser creado. - Si la hora elegida es mayor a la hora actual, un evento para el día actual.	Aprobada.
P3	Precondición: Crear un nuevo usuario en la administración de Firebase. - Ingresar las credenciales del nuevo usuario, en la pantalla de inicio de sesión, e iniciar el proceso de ingreso.	El ingreso será rechazado señalando que el usuario no tiene dispositivos asignados.	Aprobada.
P4	Precondición: Prueba P3. Actualizar permisos AWS para otorgar acceso a un dispositivo. - Ingresar las credenciales del	El ingreso es exitoso y se despliega la información correspondiente al dispositivo.	Aprobada.

ID	Descripción	Resultados Esperados	Resultados Obtenidos
	usuario, en la pantalla de inicio de sesión, e iniciar el proceso de ingreso.		
P5	Precondición: Iniciar sesión. - Partiendo del calentador de agua en estado apagado forzar manualmente su encendido. Esperar un tiempo y forzar manualmente el apagado.	Al ingresar a la vista de consumo de energía eléctrica el gráfico deberá presentar información para el rango de tiempo en el que el calentador de agua permaneció encendido.	Aprobada.
P6	Precondición: Prueba P2. - En la interfaz gráfica, presionar por tiempo prolongado el evento de encendido. - En el diálogo que aparecerá producto de la acción anterior, elegir la opción de repetir, seleccionar los días de la semana deseados y confirmar la selección.	El listado de eventos reflejará los días seleccionados en el diálogo.	Aprobada.

ID	Descripción	Resultados Esperados	Resultados Obtenidos
P7	Precondición: Prueba P2. - Deslizar el componente gráfico que presenta la información del evento de encendido hacia la izquierda y seleccionar la opción de deshabilitar.	El componente gráfico volverá a cerrarse y se desplegará un texto informativo indicando que el evento se encuentra deshabilitado. Al deslizar nuevamente el componente deberá haber la opción para volver a habilitar el evento. Seleccionarla removerá el texto informativo que apareció antes.	Aprobada.
P8	Precondición: Prueba P2. - Deslizar el componente gráfico que presenta la información del evento de encendido hacia la izquierda y seleccionar la opción de eliminar.	El evento deberá desaparecer del listado.	Aprobada.
P9	Precondición: Iniciar sesión. Tener eventos creados por otro usuario que no sea el utilizado para iniciar sesión. - Deslizar el componente gráfico que presenta la información de un	Las opciones de editar y eliminar no deberán desencadenar ninguna acción ni mostrar señales gráficas de respuesta al tacto.	Aprobada.

ID	Descripción	Resultados Esperados	Resultados Obtenidos
	evento de encendido, perteneciente a un usuario distinto al utilizado para iniciar sesión, hacia la izquierda e intentar seleccionar la opción de eliminar o editar.		

4. Conclusiones y Recomendaciones

4.1. Conclusiones

Las aplicaciones del Internet de las cosas, y los dispositivos involucrados en ellas, son realmente diversas. Por esta razón no es posible señalar inequívocamente a un protocolo de comunicación que resulte adecuado para toda ocasión.

En el caso del proyecto expuesto en este documento, se consideró propicia la utilización de MQTT en vista que: provee características de mensajería robustas, procurando consumir una porción pequeña de ancho de banda; los datos transmitidos son homogéneos y, en su mayoría, son realmente utilizados cuando son recibidos; y cuenta con considerable adopción por lo que es posible encontrar varias herramientas y librerías desarrolladas en torno al protocolo.

Se constató que al momento de exponer información mediante un API es importante considerar las necesidades de consumo de los clientes que van a interactuar con la interfaz y, en base a esto, apoyarse en los patrones de comunicación adecuados (ya sea mensajería como con publicación/suscripción o algo más tradicional como petición/respuesta).

Para el proyecto, exponer datos sobre el estado del calentador de agua mediante publicación/suscripción es apropiado debido a que es información que los consumidores siempre desean obtener, de forma actualizada. Por otro lado, establecer aquel estilo de interacción bidireccional no es adecuado para la información sobre consumo de energía eléctrica, en vista que son datos que el cliente requiere únicamente en momentos puntuales. Por ese motivo, petición/respuesta resultó ideal para el segundo caso.

Se evidenció que es relativamente sencillo lograr conectividad remota con dispositivos pensados para ser utilizados dentro del hogar. En vista de esto, la

seguridad es un aspecto importante a la hora de construir soluciones del Internet de las cosas. Si bien en el caso específico del proyecto no se maneja información sensible ni se mantienen conexiones con otros dispositivos del hogar, un dispositivo o servicio vulnerable tiene el potencial de ser el punto de entrada para comprometer este tipo de datos.

La popularidad actual de productos de software móvil, sumada a la adopción de estrategias de negocio enfocadas al rápido lanzamiento al mercado, han propiciado la aparición de plataformas que ofertan *backend* como un servicio, como Firebase. Se pudo comprobar que su utilización es ideal para enfocarse en el desarrollo del aplicativo cliente y para ocasiones en las que probar una nueva idea de producto de manera rápida, y poco costosa, es importante. Emplear uno de estos servicios permite liberarse de preocupaciones propias de mantener un *backend*; por ejemplo, la escalabilidad horizontal es dependiente del proveedor y no es difícil de conseguir.

Se comprobó que la metodología experimental planteada es idónea para sistemas que interactúan con hardware como microcontroladores. El uso de prototipos permite explorar tanto requerimientos como opciones de diseño para el dispositivo, con datos que una simulación no podría proveer.

4.2. Recomendaciones

Al momento de seleccionar una estrategia de comunicación, analizar adecuadamente cada caso de uso particular para determinar cuál de los diversos protocolos aplicables al Internet de las cosas proporciona el conjunto de características que mejor se adaptan a la funcionalidad que se desea obtener.

Utilizar certificados X.509 para la identificación de elementos con tendencia a ser estáticos dentro de la arquitectura, como microcontroladores. Es un mecanismo de encriptación e identificación ampliamente utilizado y es posible almacenarlos dentro del dispositivo, tomando en cuenta que, a diferencia de los clientes, el número de elementos está bajo el control del implementador del sistema.

En lo que respecta a la utilización de *backend as a service* es importante identificar si las funciones que dependen del servicio de un tercero, son estratégicas para la solución o un simple extra. En caso de tratarse de funcionalidades clave, es recomendable pensar en un plan de migración para cuando surja la necesidad de realizar modificaciones importantes al servicio consumido o exista algún inconveniente con él (eliminación, por ejemplo).

En cuanto a hardware (microcontroladores), para la fase de desarrollo, o de realización de prototipos, es importante seleccionar una plataforma flexible para poder acomodarse sin inconveniente a los cambios que puedan requerirse.

5. Referencias

Aparicio, D. (2015). *Objetos inteligentes: cuando la domótica evolucionó hacia el Internet de las cosas*. Recuperado el 17 de Julio de 2016 de http://www.20minutos.es/noticia/2560747/0/domotica/internet-de-las-cosas/futuro/.

AWS IoT Developer Guide. (s.f.). Recuperado el 23 de Junio de 2016 de http://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html.

Banks, A., Gupta, R. (Eds.). (2014). *MQTT Version 3.1.1*. Recuperado el 23 de Junio de 2016 http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

Barnes, B. y Fulford, G. (2008). *Mathematical Modelling with Case Studies: A Differential Equations Approach using Maple and MATLAB*. (2.ª ed.). Boca Raton, Estados Unidos: Chapman and Hall/CRC.

Carbou, R., Diaz, M., Exposito, E. y Roman, R. (2013). *Digital Home Networking*. Recuperado el 03 de Noviembre de 2014 de http://www.eblib.com.

Cohn, M. (2005). *Agile Estimating and Planning*. New Jersey, Estados Unidos: Prentice Hall.

Dodig-Crnkovic, G. *Scientific Methods in Computer Science*. Recuperado el 30 de Noviembre de 2014 de http://www.mrtc.mdh.se/publications/0446.pdf.

Fette, I., Melnikov, A. (2011). *The Websocket Protocol*. Recuperado el 09 de Julio de 2016 de https://tools.ietf.org/html/rfc6455.

Getting Started with the Arduino Yún. (s.f.). Recuperado el 23 de Junio de 2016 de https://www.arduino.cc/en/Guide/ArduinoYun.

Kyas, O. (2013). *How to: Smart Home*. Recuperado el 03 de Noviembre de 2014 de http://www.openremote.com/wp-content/uploads/2013/12/How-To-Smart-Home-PDF-OR.pdf.

Lampkin, V., Tat Leong, W., Olivera, L., Rawat, S., Subrahmanyam, N. y Xiang, R. (2012). *Building Smarter Planet Solutions with MQTT and IBM*

McEwen, A. Y Cassimally, H. (2013). *Designing the Internet of Things*. Reino Unido: Wiley.

Reddy, M. (2011). *API Design for C++*. Burlington, Estados Unidos: Morgan Kaufmann.

RMS Voltage Tutorial. (s.f.). Recuperado el 03 de Julio de 2016 de http://www.electronics-tutorials.ws/accircuits/rms-voltage.html.

Romano, Z. (2013). *Welcome Arduino Yún – The First Board Combining Arduino with Linux*. Recuperado el 23 de Junio de 2016 de https://blog.arduino.cc/2013/05/18/welcome-arduino-yun-the-first-member-of-a-series-of-wifi-products-combining-arduino-with-linux/.

Schwartz, M. (2014). *Arduino WiFi Power Switch & Energy Monitoring Device*. Recuperado el 11 de Julio de 2016 de https://www.openhomeautomation.net/arduino-wifi-switch/.

Steiner, H. (2009). *Firmata: Towards making microcontrollers act like extensions of the computer*. Recuperado el 23 de Junio de 2016 de http://www.nime.org/proceedings/2009/nime2009_125.pdf.

Waher, P. (2015). *Learning Internet of Things*. Birmingham, Reino Unido: Packt Publishing.

WebSphere MQ Telemetry. Recuperado el 23 de Junio de 2016 http://www.redbooks.ibm.com/redbooks/pdfs/sg248054.pdf.

What is Arduino?. (s.f.). Recuperado el 23 de Junio de 2016 de https://www.arduino.cc/en/Guide/Introduction.

Williams, C. (2014). *Layers of a Nodebot*. Recuperado el 23 de Junio de 2016 de https://github.com/nodebots/nodebots.io/wiki/Layers-of-a-Nodebot.

ANEXOS

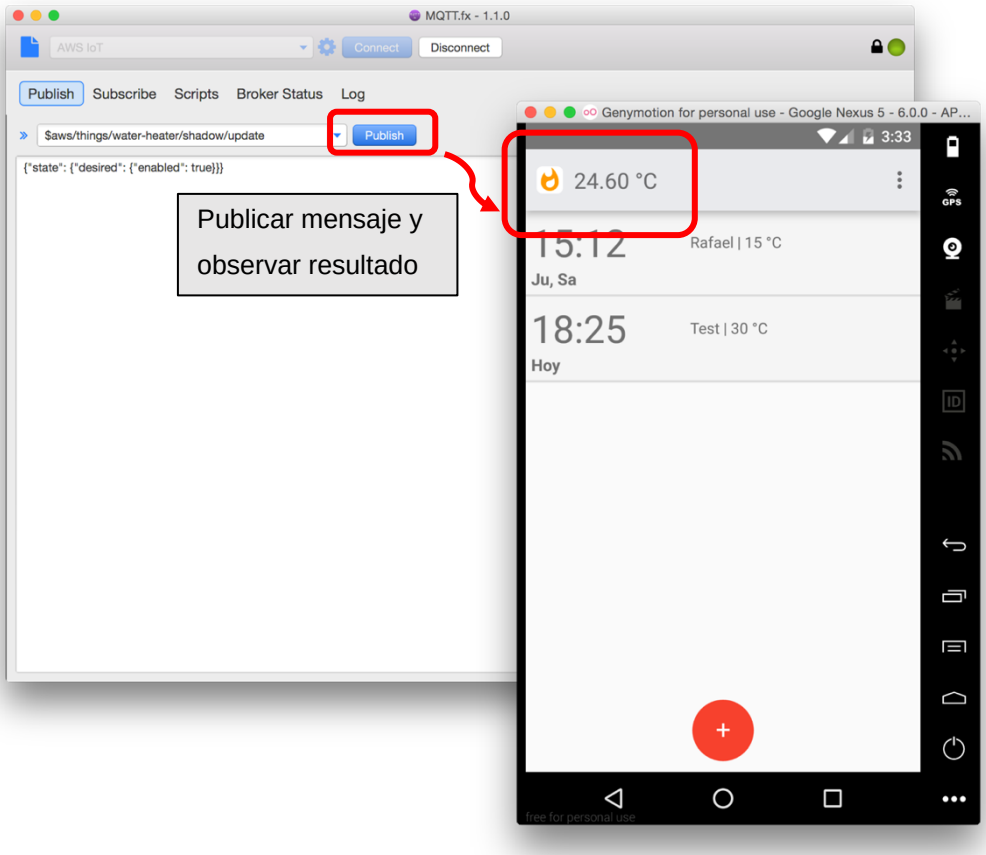
Anexo 1. Resultados Pruebas de Aceptación

Pruebas especificadas en la Tabla 7.

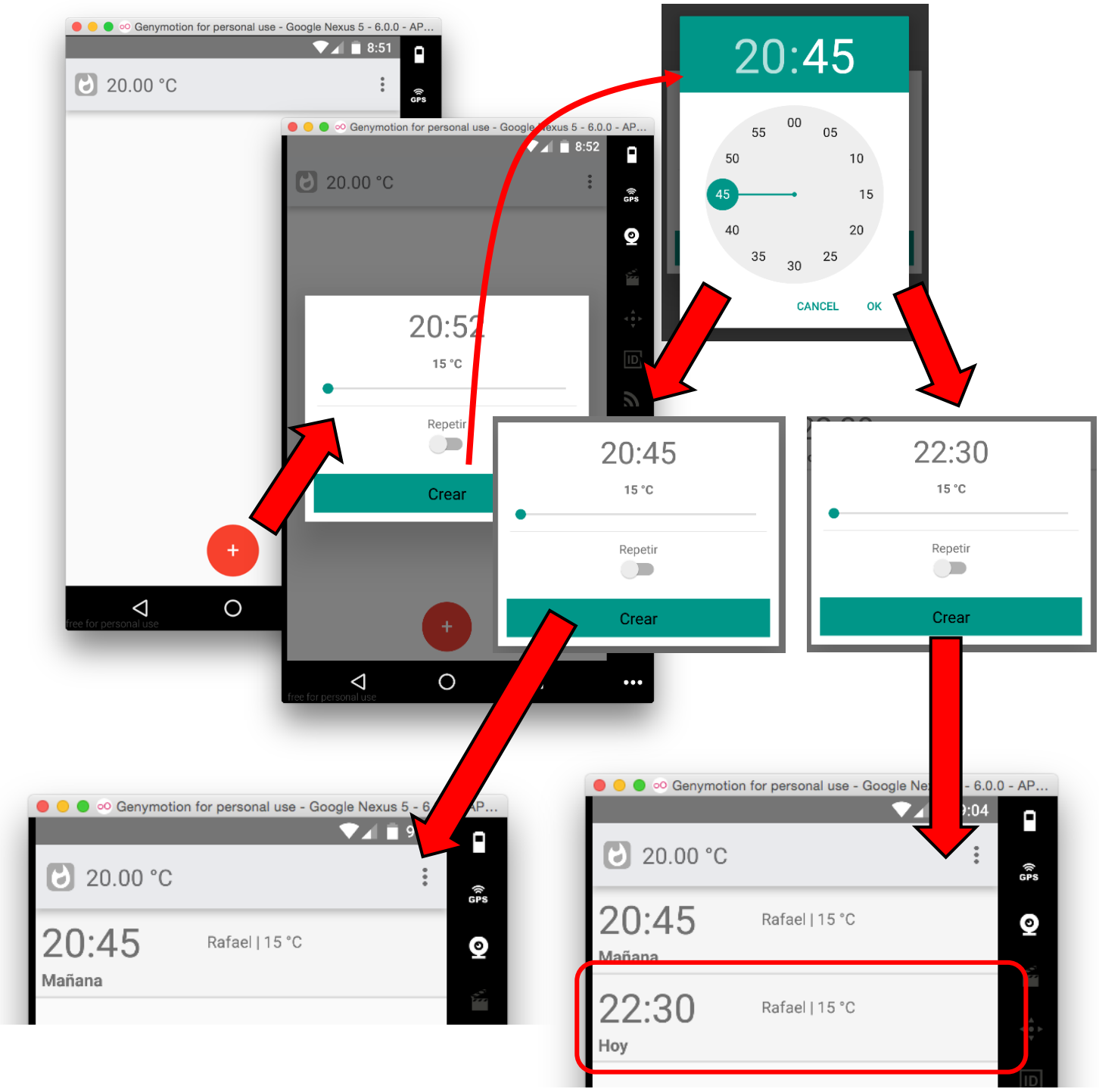
1. Prueba P1

Calentador de agua apagado:

Forzado de encendido, mediante cliente MQTT, y cambio de estado en la interfaz gráfica:



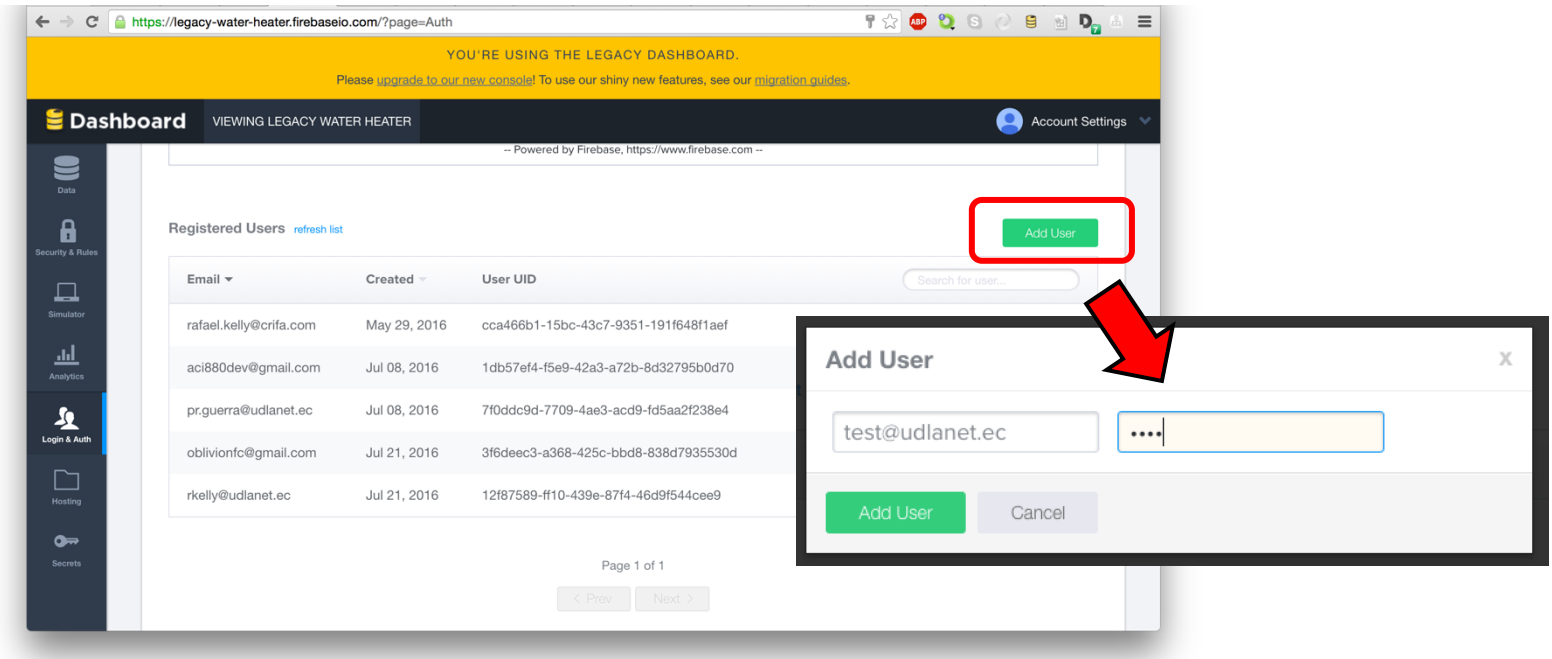
2. Prueba P2


Genymotion for personal use - Google Nexus 5 - 6.0.0 - AP...
8:51
20.00 °C
GPS
Genymotion for personal use - Google Nexus 5 - 6.0.0 - AP...
8:52
20.00 °C
20:52
15 °C
Repetir
Crear
20:45
00
05
10
15
20
25
30
35
40
45
50
55
CANCEL
OK
20:45
15 °C
Repetir
Crear
22:30
15 °C
Repetir
Crear
Genymotion for personal use - Google Nexus 5 - 6
20.00 °C
20:45
Rafael | 15 °C
Mañana
Genymotion for personal use - Google Ne
- 6.0.0 - AP...
20.00 °C
20:45
Rafael | 15 °C
Mañana
22:30
Rafael | 15 °C
Hoy

Selección de hora de encendido menor (izquierda) y mayor (derecha) a la hora actual.

3. Prueba P3

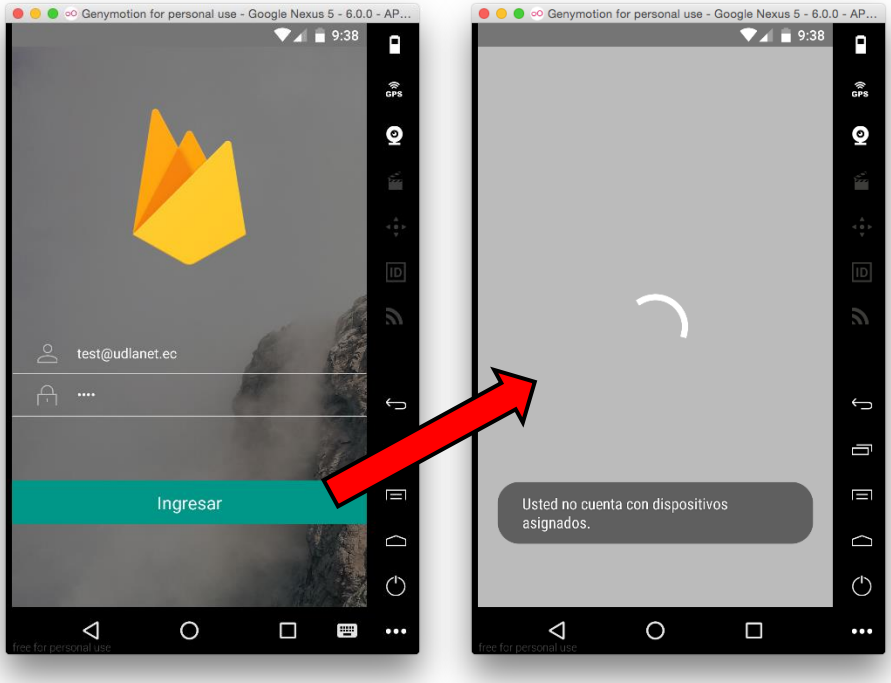
Registrar nuevo usuario en Firebase (precondición):

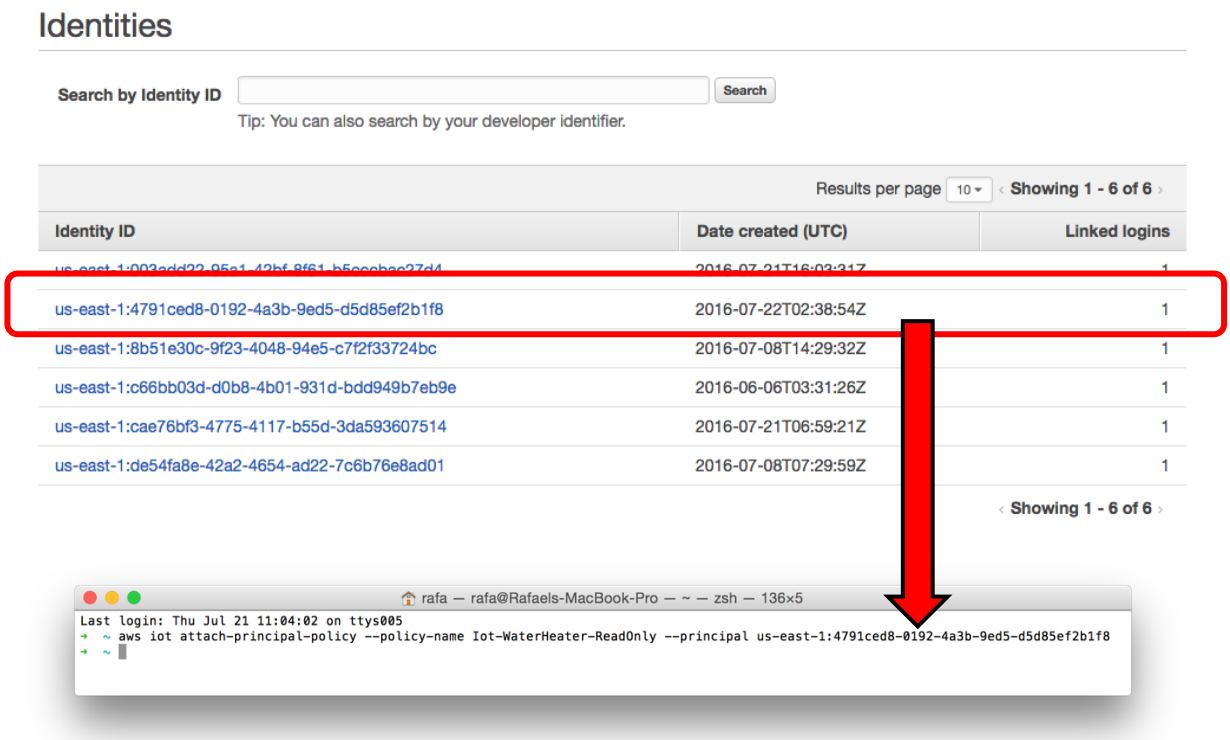


Intentar ingresar con las nuevas credenciales:

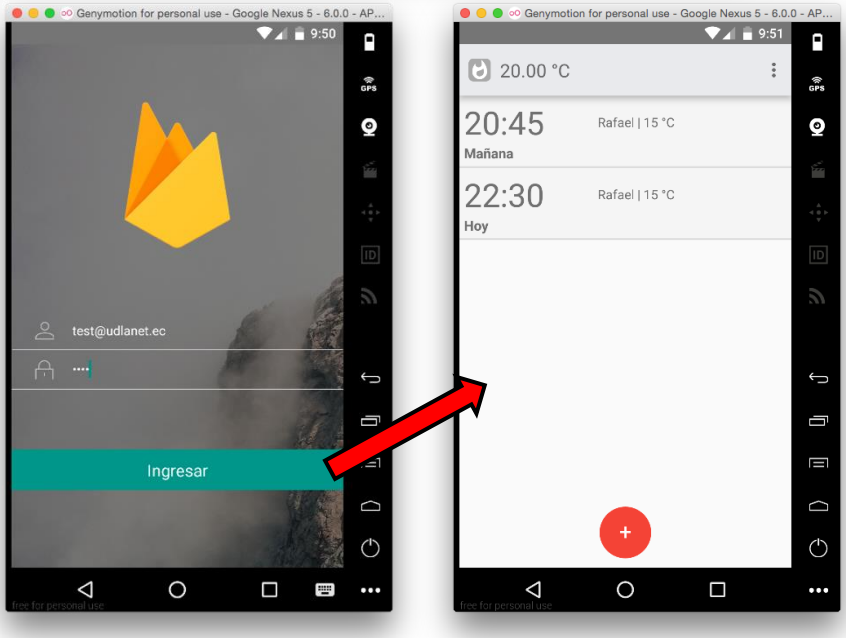
4. Prueba P4

Identificar nueva identidad en AWS y asignar permisos (precondición):

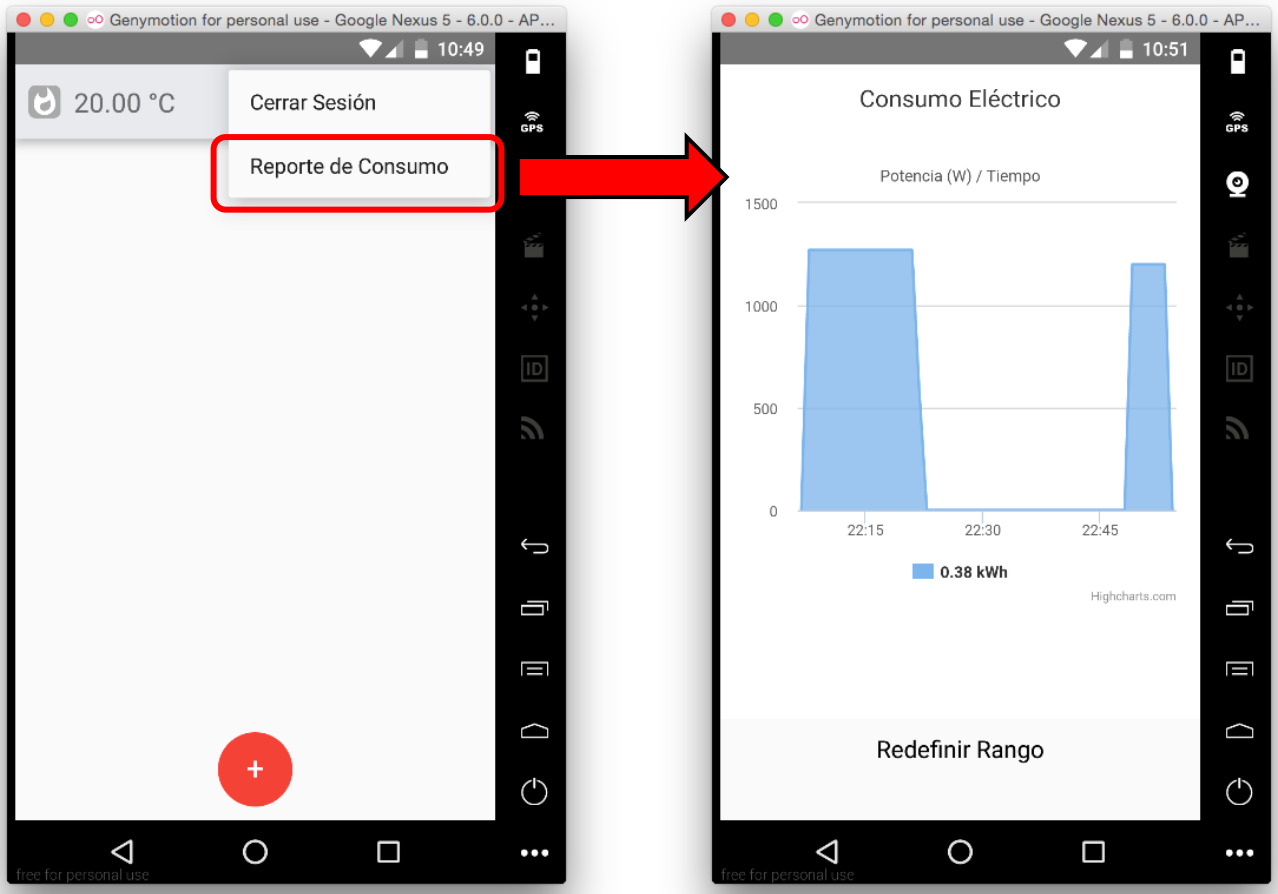




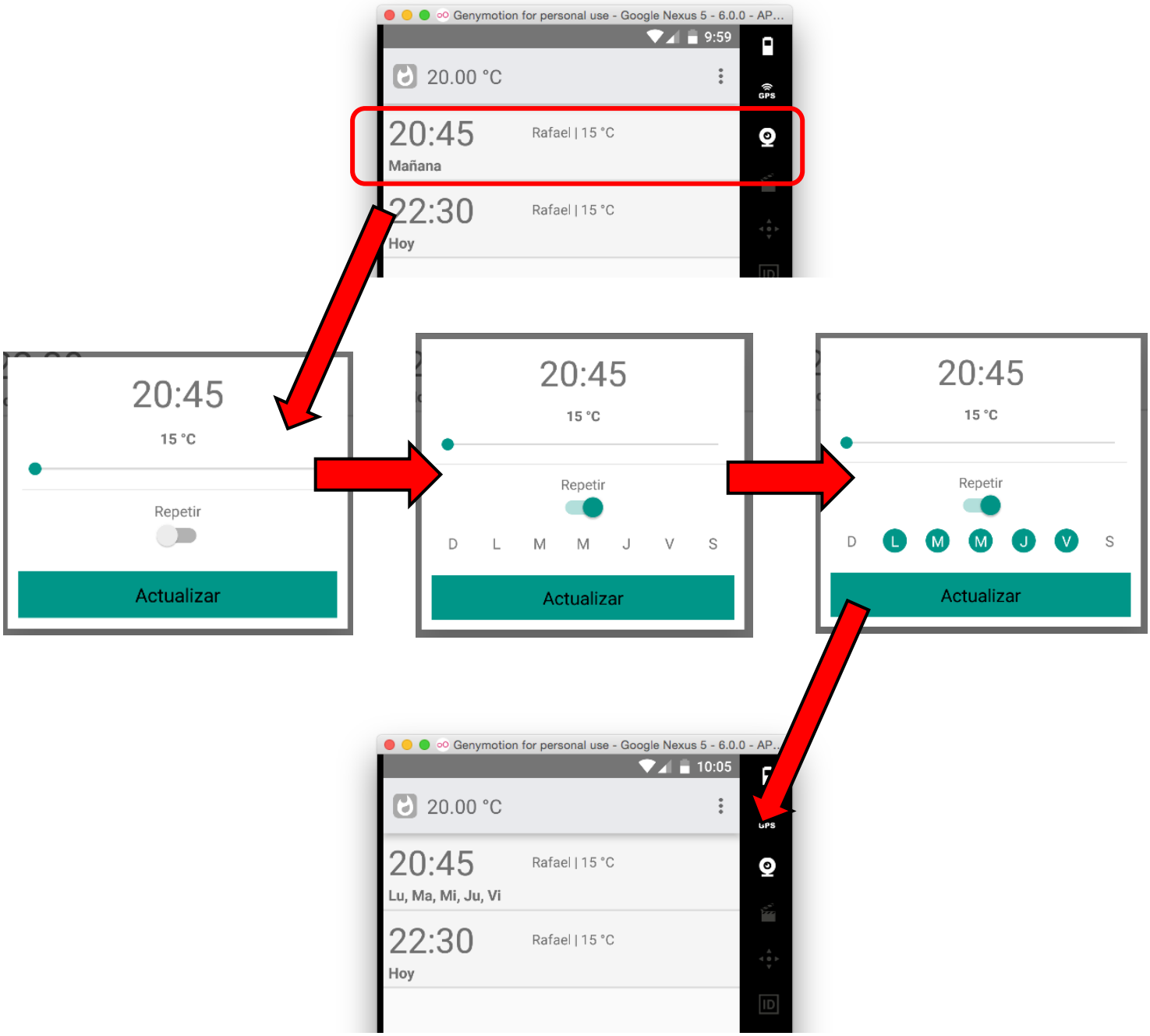
Ingresar exitosamente con las nuevas credenciales:



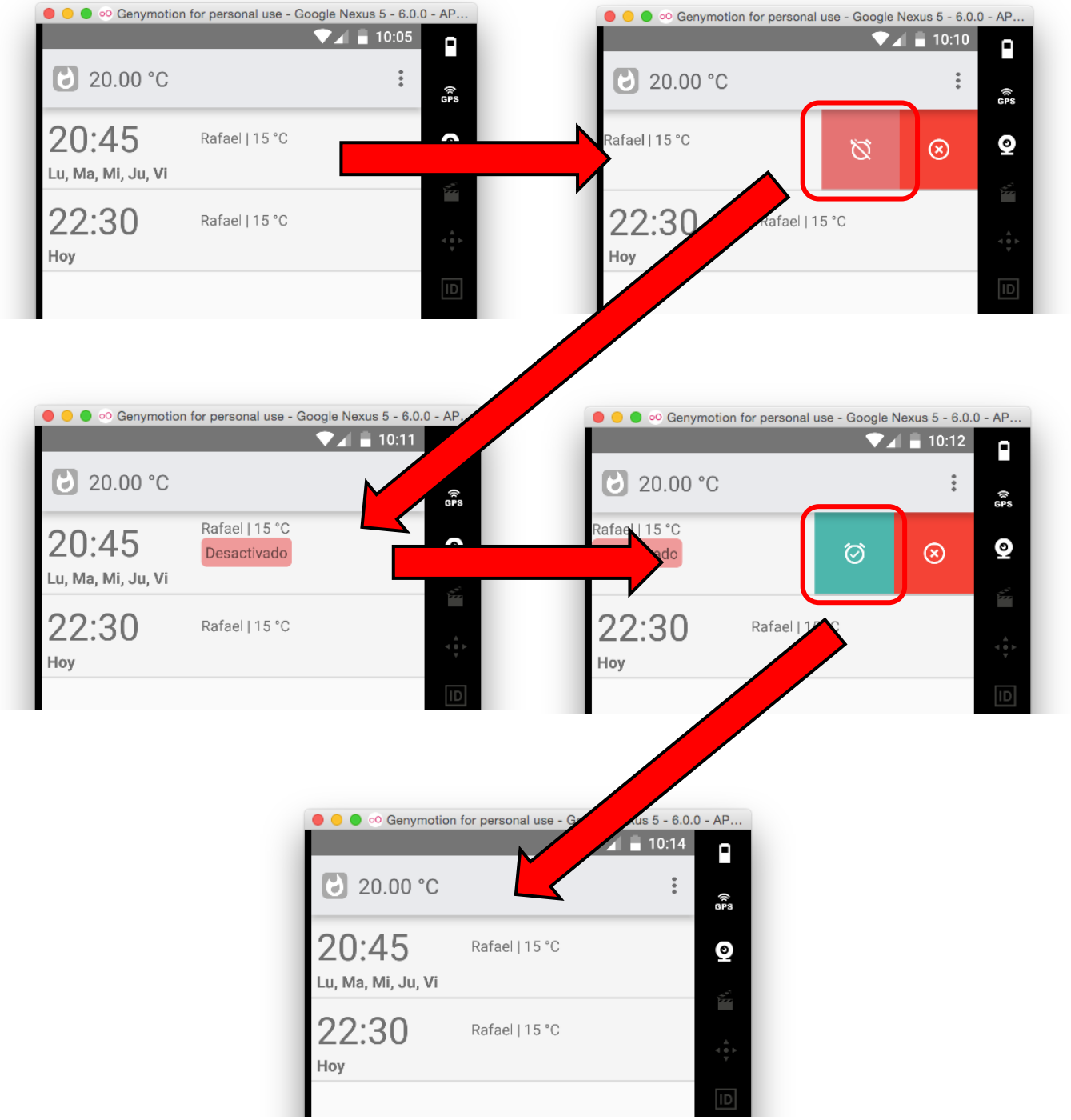
5. Prueba P5


Genymotion for personal use - Google Nexus 5 - 6.0.0 - AP...
10:49
20.00 °C
Cerrar Sesión
Reporte de Consumo
free for personal use
Genymotion for personal use - Google Nexus 5 - 6.0.0 - AP...
10:51
Consumo Eléctrico
Potencia (W) / Tiempo
1500
1000
500
0
22:15
22:30
22:45
0.38 kWh
Highcharts.com
Redefinir Rango
free for personal use

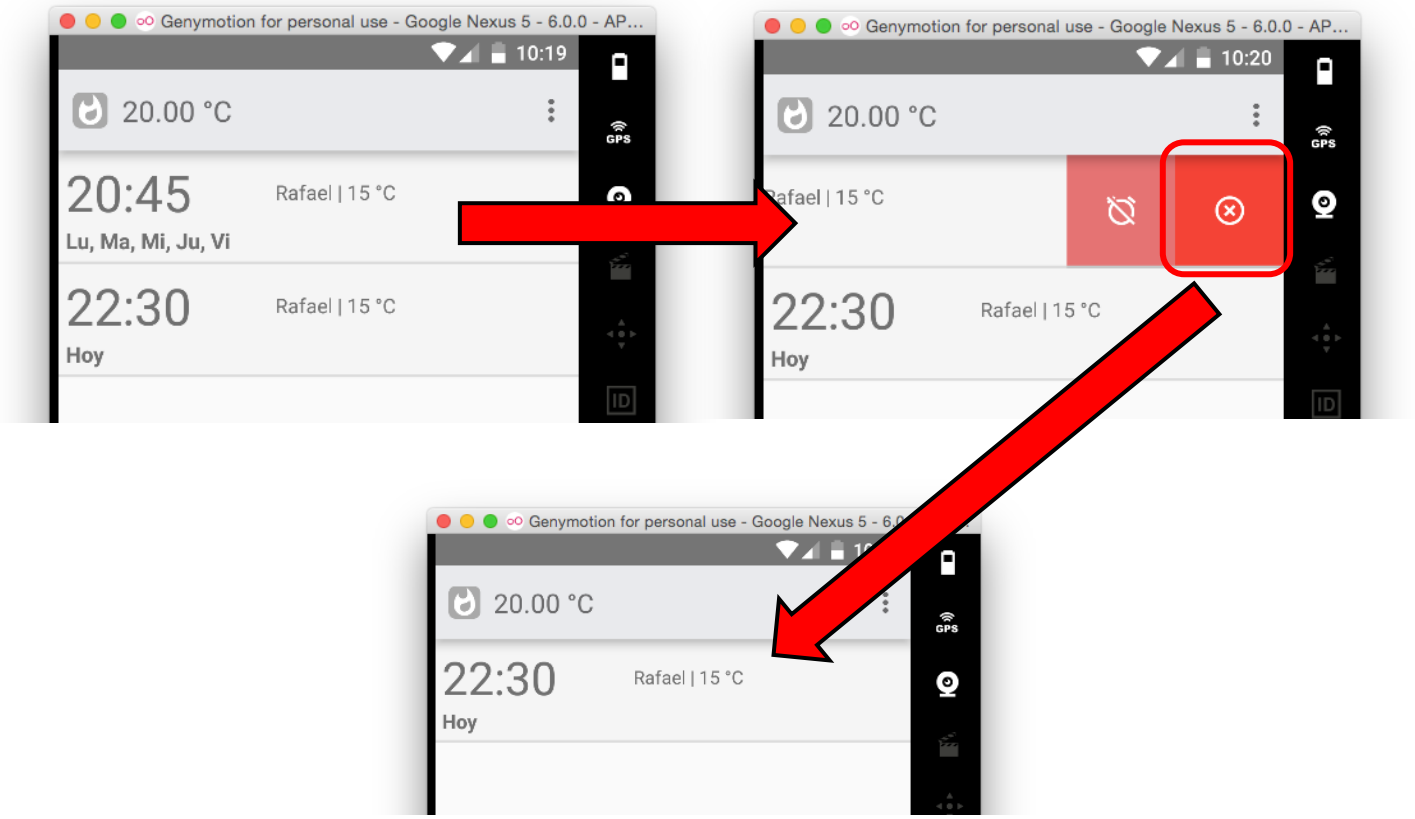
6. Prueba P6


Genymotion for personal use - Google Nexus 5 - 6.0.0 - AP...
9:59
20.00 °C
20:45
Rafael | 15 °C
Mañana
22:30
Rafael | 15 °C
Hoy
20:45
15 °C
Repetir
Actualizar
20:45
15 °C
Repetir
D L M M J V S
Actualizar
20:45
15 °C
Repetir
D L M M J V S
Actualizar
Genymotion for personal use - Google Nexus 5 - 6.0.0 - AP...
10:05
20.00 °C
20:45
Rafael | 15 °C
Lu, Ma, Mi, Ju, Vi
22:30
Rafael | 15 °C
Hoy

7. Prueba P7


Genymotion for personal use - Google Nexus 5 - 6.0.0 - AP...
10:05
20.00 °C
20:45
Rafael | 15 °C
Lu, Ma, Mi, Ju, Vi
22:30
Rafael | 15 °C
Hoy
Genymotion for personal use - Google Nexus 5 - 6.0.0 - AP...
10:10
20.00 °C
Rafael | 15 °C
22:30
Rafael | 15 °C
Hoy
Genymotion for personal use - Google Nexus 5 - 6.0.0 - AP...
10:11
20.00 °C
20:45
Rafael | 15 °C
Desactivado
Lu, Ma, Mi, Ju, Vi
22:30
Rafael | 15 °C
Hoy
Genymotion for personal use - Google Nexus 5 - 6.0.0 - AP...
10:12
20.00 °C
22:30
Hoy
Genymotion for personal use - Google Nexus 5 - 6.0.0 - AP...
10:14
20.00 °C
20:45
Rafael | 15 °C
Lu, Ma, Mi, Ju, Vi
22:30
Rafael | 15 °C
Hoy

8. Prueba P8



9. Prueba P9

Para eventos no creados por el usuario, las opciones de eliminar y deshabilitar no realizarán ninguna acción.