

TRABAJO FINAL DE INGENIERÍA INFORMÁTICA



*Pontificia Universidad Católica Argentina
"Santa María de los Buenos Aires"
Facultad de ciencias fisicomatemáticas e ingeniería*

ALUMNO Facundo Spagnuolo

TUTOR Ricardo Di Pasquale

TEMA Domótica

FECHA 21 de Julio de 2016

INDICE

Abstract	3
Introducción	5
<i>Domótica</i>	5
<i>Qué nos motivó?</i>	6
<i>Cómo lo llevamos a cabo?</i>	7
Arquitectura	9
Dispositivos Cliente	11
<i>Introducción</i>	11
<i>Python</i>	11
<i>Raspberry</i>	13
<i>Implementación</i>	15
<i>Configuración</i>	34
Servidor	38
<i>Introducción</i>	38
<i>Ruby</i>	39
<i>Paradigma OO</i>	41
<i>TDD</i>	43
<i>Rails</i>	46
<i>PostgreSQL</i>	49
<i>Implementación</i>	50
Protocolos	65
<i>Introducción</i>	65
<i>JSON</i>	65
<i>MQTT</i>	92
Conclusión	96
Bibliografía	98



Abstract

La tecnología avanza enormemente y el mundo está atento a las nuevas novedades y tendencias. La tecnología aplicada al hogar, conocida como domótica, integra automatización, informática y nuevas tecnologías de comunicación. Todas ellas dirigidas a mejorar la comodidad, la seguridad y, en definitiva, el bienestar dentro de los hogares.

El intenso avance tecnológico experimentado en los últimos años ha contribuido mucho al desarrollo de la domótica en aspectos tan cotidianos como la iluminación, climatización, seguridad, comunicación, entre otros.

Se inició este proyecto con el incentivo de aportar una implementación para la automatización de hogares. Basados en la investigación respecto al tema mencionado, haciendo uso de algunos conceptos de las metodologías ágiles y aplicando tecnologías de punta, logramos IntelliHome. Un sistema que permite controlar distintos artefactos de un hogar a través de nuestras computadoras o dispositivos móviles.

Se caracteriza por ser seguro, ya que sólo aquellos usuarios autorizados y autenticados son los que pueden manipular la aplicación. Está al alcance de la mano dado que puede ser manejable desde cualquier dispositivo de manera remota a través aplicaciones web o mobile. Es también expansible en funcionalidad, por que existen muchas otras áreas y artefactos explorables de un hogar que todavía no han sido explotadas, y que pueden ser cubiertas y manejables a través de IntelliHome.

Pudimos lograr mantener la idea de implementar el sistema con costos bajos. Sin demasiada inversión, y sin necesidad de comprar artefactos específicos sino que buscamos la manera de poder adaptar y reutilizar aquellos que ya tenemos en nuestros hogares.

Particularmente este trabajo se centra en la implementación del backend de IntelliHome, es decir, todo lo que sucede detrás de escena. Esto implica la planificación, diseño y desarrollo de toda la arquitectura necesaria para la implementación.

Se llevó a cabo la instalación y programación de los dispositivos requeridos para controlar los distintos artefactos del hogar. También el desarrollo del servidor encargado de gestionar dichos dispositivos de manera eficiente, y de recordar el estado de nuestro sistema. Además, es quien nos abstrae de la compleja arquitectura y nos brinda una manera fácil de comunicarnos con los artefactos de nuestro hogar de una manera sencilla.

A su vez nos brinda la posibilidad de adaptarlo, ajustándolo según nuestras necesidades, logrando optimizarlo, y haciéndolo eficiente de tal forma que nos ahorre el trabajo de realizar ciertas labores delegándole esa responsabilidad al sistema.

Por último, y lo más interesante, es que nos otorga la facilidad de poder manejarlo y gestionarlo de manera remota. Expone una interfaz que puede ser explotada por distintas aplicaciones web o mobile para ser controlado por sus usuarios a través de sus computadoras o dispositivos móviles.

Introducción

Domótica

En estos últimos años se han estado produciendo cambios importantes e innovadores en distintos ambientes y aspectos que acechan por sobre todas las cosas, al entorno de los seres humanos. No sólo a nuestros momentos de ocio o ecosistemas laborales, sino también a los lugares en los que residimos. Estos cambios tienen un impacto altamente positivo en nuestra comodidad y seguridad, brindándonos confort, manteniéndonos comunicados, y permitiéndonos automatizar y controlar de manera remota múltiples artefactos hogareños.

Se llama domótica al conjunto de sistemas capaces de automatizar una vivienda, aportando servicios de gestión energética, seguridad, bienestar y comunicación, y que pueden estar integrados por medio de redes interiores y exteriores de comunicación, cableadas o inalámbricas, y cuyo control goza de cierta ubicuidad, desde dentro y fuera del hogar. El término domótica viene de la unión de las palabras **domus** (que significa *casa* en latín) y **tica** (de *automática*, palabra en griego, *que funciona por sí sola*). ¹

Generalmente, la mayor parte de los aspectos relacionados con la domótica no son exclusivos del hogar ni han sido innovados especialmente para ellos, sino que suelen provenir de la aplicación en fábricas, hoteles u oficinas. Ambientes donde la automatización de ciertos aspectos implica de manera directa, la eficiencia del negocio.

Además, los sistemas de domótica suelen ayudarnos a lograr un mayor ahorro energético, ya que hacen un uso más eficiente de los diferentes artefactos a nuestra

¹ Definición de domótica - Wikipedia

disposición, sin necesidad de sustituir los aparatos o sistemas del hogar por otros que consuman menos energía.

También nos aportan a los esquemas de climatización, en todo lo que es programación y zonificación. Control de toldos y persianas eléctricas.

Automatización de sistemas de iluminación. Sistemas de seguridad con cámaras IP y alarmas de intrusión, simuladores de presencia.

Por sobre todas las cosas es muy importante el concepto de las comunicaciones con los sistemas o infraestructuras que posee el hogar, garantizando ubicuidad en el control tanto externo como interno, de manera remota a través de internet.

El concepto diseño para todos es un movimiento que pretende crear la sensibilidad necesaria para que al diseñar un producto o servicio se tengan en cuenta las necesidades de todos los posibles usuarios, incluyendo las personas con diferentes capacidades o discapacidades, es decir, favorecer un diseño accesible para la diversidad humana. El objetivo no es que las personas con discapacidad puedan acceder a estas tecnologías, sino que el objetivo de estas tecnologías es favorecer la autonomía personal.

Qué nos motivó?

La domótica no es un área nueva en la rama de la tecnología, pero si algo poco explorado en nuestro país. En Argentina la domótica surge a comienzos de la década de 1990, de la mano de empresas de tecnología que incorporan el concepto y lo desarrollan. Estas empresas comienzan a hablar de domótica refiriéndose a las casas del futuro, realizando algunas aplicaciones parciales, participando en ferias y notas periodísticas que colaboraron con la difusión del concepto. En el año 2007 se realizó la primera exposición exclusiva de domótica y el primer congreso en Córdoba.

Tal es así, que la domótica en nuestro país, no ha sido explotada de manera abarcadora en todas sus aristas, y mucho menos logró estar al alcance cotidiano, principalmente debido a sus costos.

Como todo informático podríamos asegurar que la idea de automatizar procesos es algo que nos llama seguidamente la atención. En gran parte porque de eso se trata nuestra tarea, aplicar nuestros conocimientos para resolver problemas a través de un sistema, a través de las computadoras, pero también nos resultaba realmente innovador y divertido.

Cómo lo llevamos a cabo?

A ser honestos, en un comienzo nos pareció algo inalcanzable, algo realmente poco explorado, fuera de lo común, y que por su puesto iba a requerir de una amplia gama de conocimientos, cubriendo distintas áreas, para que lográramos una implementación de punta a punta. Necesitábamos de conocimientos de programación de software, programación de dispositivos electrónicos, redes, seguridad, sistemas distribuidos, interfaces, bases de datos y muchos más.

Pero una vez, nuestra inquietud por explorar algo nuevo, y el hecho de aprovechar el tiempo que debíamos dedicarle a un trabajo final, ganó de nuevo. Decidimos investigar y contar con la ayuda de gente especializada que nos pudiera orientar en nuestras incertidumbres.

Por lo tanto, decidimos empezar de a poco, lograr apenas una primera implementación. Comenzamos investigando sobre cómo podríamos encender una bombita de luz utilizando una computadora. Lo único que hayamos fueron implementaciones antiguas utilizando puertos seriales, que hoy en día ya ni se fabrican.

Continuamos informándonos y hallamos unas plaquetas programables que nos podrían servir para manejar ciertos voltajes con el uso de relés ².

Es así que decidimos realizar un trabajo de manera iterativa e incremental, tomando como práctica algunos conceptos de las metodologías ágiles. Trabajamos con iteraciones, aportando un valor agregado en cada una de ellas, realizando demostraciones entre nosotros mismos y otras personas interiorizadas con el tema.

Comenzamos a recolectar cada vez más feedback y fuimos perfeccionando y ampliando las funcionalidades de IntelliHome. Comenzamos a estar cada vez más sumergidos en los temas de domótica, y por sobre todas las cosas, en contacto con otras personas que también dedicaban su tiempo a investigar sobre este mundo.

² El relé o relevador es un dispositivo electromagnético. Funciona como un interruptor controlado por un circuito eléctrico en el que, por medio de una bobina y un electroimán, se acciona un juego de uno o varios contactos que permiten abrir o cerrar otros circuitos eléctricos independientes.

Arquitectura

Desde el punto de vista de donde reside la inteligencia del sistema de domótica, existen diferentes tipos de arquitecturas posibles.

Existen arquitecturas centralizadas, basadas en un controlador central encargado de recibir información de múltiples sensores, que luego es procesada para realizar las acciones que sean deseadas en el hogar por medio de actuadores.

Otra arquitectura posible es una distribuida, donde toda la inteligencia del sistema está distribuida en distintos módulos. Estos módulos poseen la suficiente inteligencia para actuar de la manera en que se necesite en base a la lectura de los sensores. Este tipo de esquemas suele ser típico con el uso de redes inalámbricas. Por último, podemos encontrar arquitecturas mixtas. Estos últimos son por un lado descentralizados en cuanto a que disponen de varios dispositivos capaces de adquirir y procesar la información de múltiples sensores, pero que luego deben transmitirlos entre varios dispositivos para llevar a cabo distintas acciones deseadas en nuestro hogar. Es decir, no son completamente independientes para decidir qué hacer en base a la información recolectada.

El backend de IntelliHome está basado en una arquitectura mixta. Dicha arquitectura fue planificada como una arquitectura cliente-servidor. Los clientes son los dispositivos encargados de realizar la lectura de distintos tipos de sensores. Pero no son los responsables de decidir qué acción realizar al respecto de las lecturas realizadas. Para ello se presenta el servidor, los clientes deben enviarle toda la información recolectada al servidor central, quién será el responsable de analizarla y tomar las decisiones para accionar sobre nuestro hogar. Más aún, este servidor central no es quien verdaderamente ejecuta esas acciones, sino que se comunica nuevamente con los dispositivos correspondientes que finalmente son los encargados de controlar distintos tipos de actuadores que finalmente controlan distintos artefactos de nuestro hogar.

Por otro lado, el servidor central no sólo posee la inteligencia del hogar, sino que además es quien nos permite comunicarnos con nuestro hogar desde Internet. Este nos abstrae de dicha arquitectura para poder comunicarnos y controlarla desde el mundo exterior (ver Figura 1):

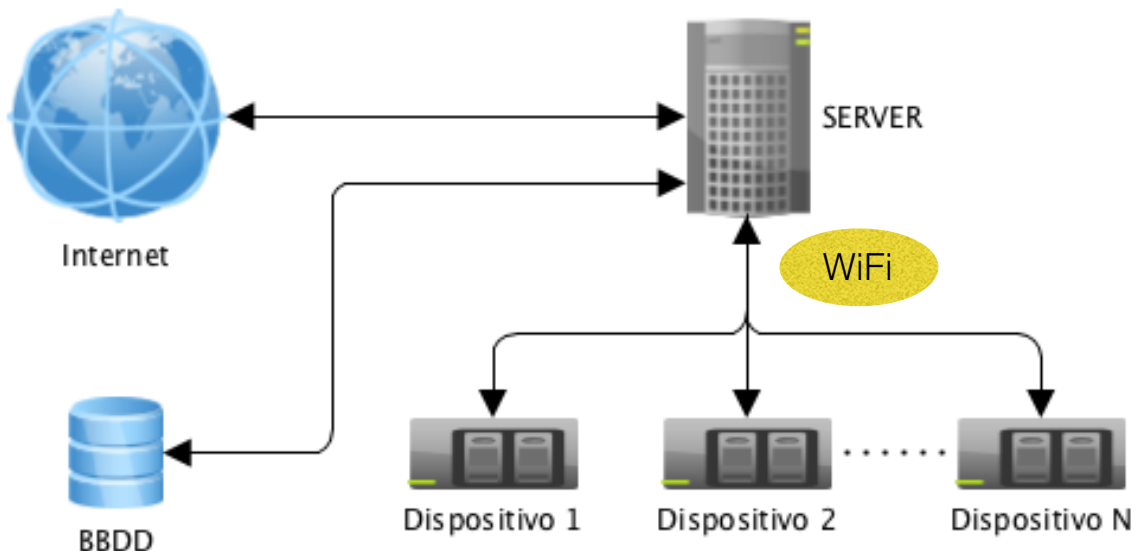


Figura 1. Diagrama de la arquitectura de IntelliHome

Los dispositivos fueron programados en Python y con la ayuda de determinadas instalaciones, nos permiten controlar artefactos eléctricos y distintos tipos de sensores dentro del hogar. Fueron pensados para ser colocados de manera inalámbrica en los distintos ambientes del hogar. De esta manera podemos controlar cada ambiente por separado, aunque su uso puede ser todavía más optimizable.

El servidor central fue programado en Ruby. Es quien gestiona la base de datos donde se almacenan los datos recolectados de los dispositivos, la información necesaria sobre los distintos ambientes del hogar, y además gestiona las credenciales de los distintos usuarios habilitados para gestionarlo. Además, se encarga de la autenticación de los usuarios que se comunican a través de Internet, brindando la seguridad de que sólo aquellos autorizados puedan controlar los distintos ambientes del hogar.

Dispositivos Cliente

Introducción

Los dispositivos cliente de IntelliHome fueron logrados con el uso de Raspberry Pi, estos son computadoras de placa simple o reducida, que permiten ser programadas en distintos lenguajes, entre ellos Python.

Las Raspberry Pi permiten ser conectadas con distintos tipos de sensores, entre ellos sensores de humedad del ambiente, sensores de temperatura del ambiente, sensores de humedad del suelo, sensores lumínicos, sensores de movimientos, y muchos otros. Con la ayuda de distintos programas realizados en Python nos es posible leer estos sensores para captar distinta información sobre el estado de los ambientes de nuestro hogar para luego comunicarla al servidor central.

Además a través de las Raspberry Pi, podemos controlar distintos actuadores, con el fin de controlar distintos artefactos de nuestro hogar tales como circuitos eléctricos o pasos de corriente de manera independiente. Estos actuadores pueden ser a su vez controlados desde Internet con la interfaz que nos provee el servidor central.

Python

Python es un lenguaje de programación interpretado con una filosofía que hace hincapié en una sintaxis que nos permita lograr sentencias legibles como si fueran oraciones escritas. Se lo considera un lenguaje de programación multiparadigma, ya que principalmente es un lenguaje orientado a objetos, pero también nos permite aplicar programación estructurada, y en menor medida, programación funcional.

Es un lenguaje interpretado, que a diferencia de los lenguajes compilados, que se basan en traducir un programa desde su descripción a código máquina del sistema, los lenguajes de programación interpretados, sólo realizan la traducción a medida que es necesaria, generalmente, instrucción por instrucción. Además normalmente no guardan el resultado de dicha traducción. Es también un lenguaje dinámicamente tipado, es decir, que una misma variable puede tomar valores de distinto tipo en distintos momentos. La mayoría de lenguajes de tipo dinámico son lenguajes interpretados.



Figura 2. Logo de Python

Python fue creado a finales de 1980 por Guido van Rossum en el Holanda. El nombre del lenguaje proviene de su afición por los humoristas británicos Monty Python. Actualmente, el lenguaje es administrado por la Python Software Foundation, una organización sin fines de lucro creada en 2011 cuya función es fomentar el desarrollo de la comunidad y es responsable del desarrollo de dicho lenguaje y la obtención de fondos.

Python es utilizado en muchos dominios, principalmente en la actualidad en lo que es el desarrollo de aplicaciones web, Python ofrece frameworks como Django y Pyramid, y además dispone de muchas librerías que implementan protocolos relacionados con Internet como sockets, HTML, JSON, XML, FTP, etc. Es también utilizado en el ámbito de la computación científica, las matemáticas y mismo en ambientes educativos.

Principalmente se decidió utilizar Python ya que resulta ser un lenguaje bastante flexible, dado que es multiparadigma, con una sintaxis bastante amena y sencilla, que nos brindó cierta flexibilidad a la hora de programar lo que fuera necesario para controlar los dispositivos Raspberry Pi.

Raspberry

Raspberry Pi es una computadora de placa reducida, también llamada computadora de placa única o computadora de placa simple. Fue desarrollada en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas logrando un dispositivo realmente interesante a bajo costo.

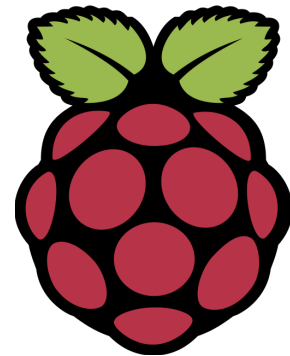


Figura 3. Logo de Raspberry Pi

Historia

En 2006 habían comenzado los primeros diseños de Raspberry Pi, y recién en mayo de 2009 fue fundada la Fundación Raspberry Pi en Caldecote, Reino Unido como una asociación caritativa regulada por la Comisión de Caridad de Inglaterra y Gales. Eben Upton, administrador de la fundación, se puso en contacto con un grupo de profesores, académicos y entusiastas de la informática con la idea de lograr una computadora con el objetivo de incentivar a los niños a que aprendieran informática. En agosto de 2011, se fabricaron las primeras cincuenta placas alpha que tenían las mismas características que el primer modelo productivo de Raspberry Pi. Eran un poco más grandes pero se lograron hacer algunas demostraciones ejecutando sistemas operativos Linux y video full HD a través de interfaces HDMI.

En diciembre de 2011, se fabricaron 25 placas beta, que fueron ensambladas y probadas. El diagrama de los componentes de las placas sería el que más tarde saldría a producción. Si bien se encontraron algunos errores de diseño, fueron corregidos y realizó una demostración que luego dio lugar a las primeras subastas. Comenzaron subastando diez placas que fueron compradas entre 3.500 y 16.300 libras.

Finalmente, a fines de febrero de 2012, se fabricó el primer lote de 10.000 placas en Taiwan y China, que fueron vendidas a través del sitio web de la fundación. Esto le

permitió a la fundación ahorrar costos de producción y para luego reinvertirlo en investigación y desarrollo.

“Seis años después del origen del proyecto, estamos cerca de finalizar el primer arranque del proyecto - aunque esto es solo el principio de la historia de Raspberry Pi”, fue el anuncio que la fundación publicó en la página para dar inicio a sus ventas. En los seis meses se habían vendido 500.000 unidades. Recién en diciembre de 2015 surgió Raspberry Pi 2, un modelo con mejores prestaciones. Y en febrero de 2016, salió a la venta el último modelo, Raspberry Pi 3.

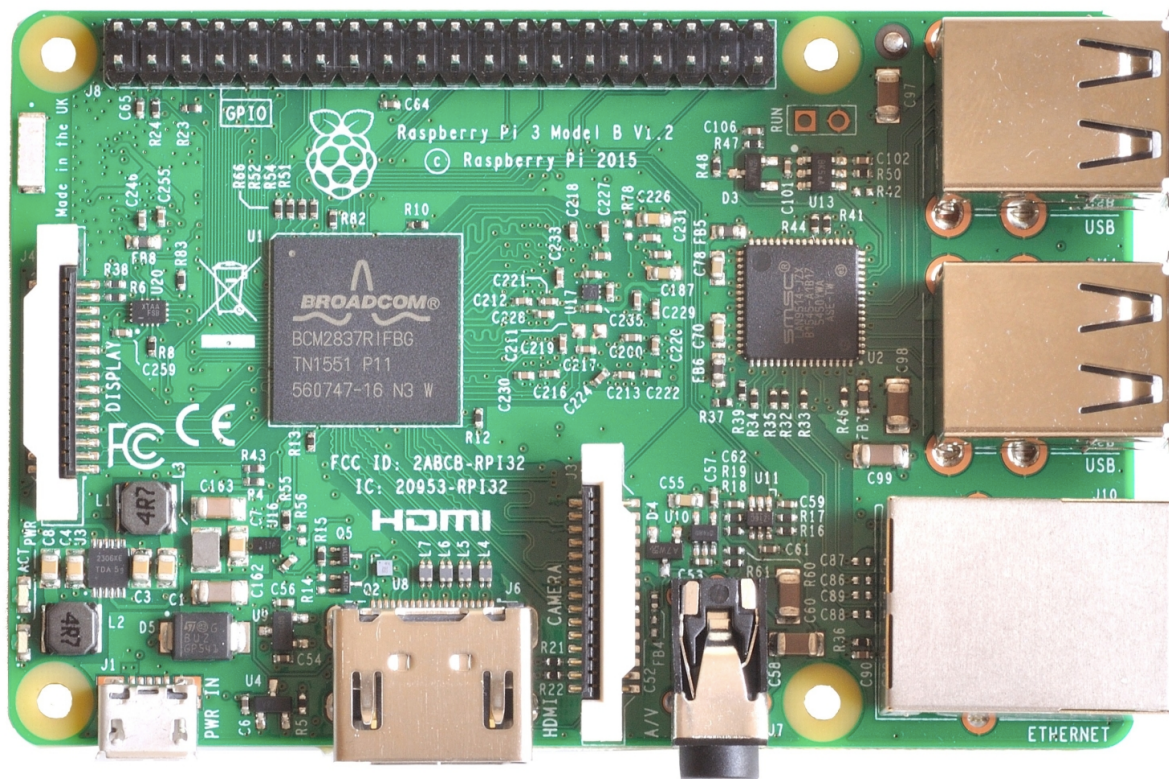
Software y Hardware

Actualmente se encuentra en la disyuntiva de si es hardware libre o posee derechos de marca, ya que si bien disponen de contratos de distribución y venta con dos empresas, cualquiera puede ser revendedor y además es de uso libre. En cambio el software sí es open source, siendo su sistema operativo oficial una versión adaptada de Debian, denominada Raspbian, aunque permite otros sistemas operativos, incluido una versión de Windows 10. La fundación es la encargada de dar soporte para realizar las descargas de sus sistemas operativos, además promueve el aprendizaje de varios lenguajes de programación soportados por la arquitectura como Python, Perl y Ruby, entre otros.

Para llevar a cabo los dispositivos necesarios para la implementación utilizamos el último modelo de Raspberry Pi, específicamente Raspberry Pi 3 Model B (ver Figura 4). Esta placa cuenta con las siguientes características:

- Procesador Broadcom 1.2GHz 64-bit quad-core
- Procesador de gráficos 3D VideoCore IV
- 1GB RAM
- 4 puertos USB
- 40 pins GPIO
- Bluetooth 4.1
- Puerto Full HDMI
- 802.11n Wireless LAN (WiFi)

- Puerto Ethernet
- Entrada de audio y video compuesto de 3.5mm
- Interface para cámara
- Slot para memoria micro SD



Implementación

Los modelos de Raspberry Pi utilizadas para la implementación de este trabajo cuentan con 40 pines GPIO, que significa *General Purpose Input/Output*, es decir, Entrada/Salida de Propósito General. Son pines genéricos, cuyo comportamiento se puede controlar o programar por el usuario en tiempo de ejecución. Este tipo de pines no tienen ningún propósito predefinido, sino que surgen con la idea de utilizarlos para controlar cosas de manera digital.

En la implementación de IntelliHome utilizamos estos pines tanto para recolectar información de distintos sensores como también para controlar los actuadores que nos permiten accionar sobre los ambientes de nuestro hogar.

Sensores

Los sensores son dispositivos que nos permiten medir distintas cualidades de interés en nuestro hogar, que luego nos permiten tomar decisiones con respecto a ellas. Para la implementación de IntelliHome se utilizaron distintos tipos de sensores que se describen a continuación:

Sensor de Humedad

La humedad es la presencia de agua en el aire, y muchas veces puede afectar distintos procesos. La cantidad de humedad puede afectar características físicas, químicas y biológicas de distintas cosas, es por eso que la medición de la humedad a través de sensores puede ser crítico para ciertas industrias.

Estos sensores suelen utilizarse para aplicación médica como respiradores, esterilizadores, incubadoras, y otros. También en procesos químicos como la purificación de gas, hornos, producción textil y de alimentos, o también en la industria agropecuaria para distintos cultivos.

Existen dos tipos de sensores de humedad, los relativos (RH) o absolutos (AH). La gran mayoría de los sensores de humedad son relativos y usan diferentes principios de sensado. Los más comunes de todos ellos son aquellos de tipo capacitivos o resistivos. Los sensores capacitivos se basan en la carga de un capacitor con un material dieléctrico higroscópico, generalmente algún tipo de plástico o polímero. Básicamente el dieléctrico permite absorber la humedad del ambiente y hacer variar la carga del capacitor. Los resistivos funcionan a través de unas resistencias especiales que varían su valor resistivo según la humedad del ambiente.

Para la implementación se utilizó el sensor de humedad DHT11 (ver Figura 5). Es un sensor de humedad capacitivo, de bajo costo, que nos permite medir la humedad

del ambiente de manera digital, es decir, que no se requiere de entradas analógicas. Por medio del uso de librerías podemos obtener el valor de humedad sentido.

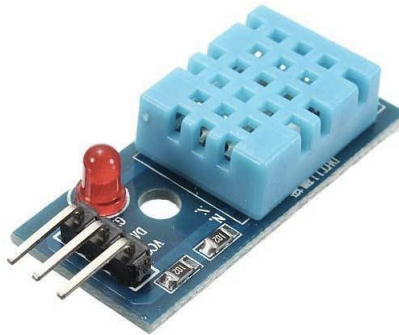


Figura 5. Sensor de humedad y temperatura DHT11

Sensor de Temperatura

La temperatura o el calor del ambiente es otro dato de interés para tener en cuenta en nuestros procesos. Suele ser tomada en cuenta para el uso de dispositivos con termostato, sistemas de calentamiento de agua doméstico, control de calderas industriales, cultivos realizados en invernaderos, etc.

Existen distintos tipos de sensores de temperatura. Por un lado están los sensores mecánicos, que son quizás los de uso cotidiano, suelen utilizar metales para indicar distintas medidas de calor debido a la dilatación o contracción de los mismos, como por ejemplo, comúnmente, el mercurio. Además existen los sensores eléctricos, que generalmente trabajan con termistores. Estos son unas resistencias especiales que presentan una variación en su resistividad, normalmente, a mayor temperatura menor resistividad.

Para la implementación utilizamos los sensores de temperatura DHT11 (ver Figura 5), ya que nos permiten medir la temperatura del ambiente aparte de la humedad del mismo. Es un sensor eléctrico que trabaja con un termistor, y que nos permite medir la temperatura de manera digital con el uso de algunas librerías particulares.

Sensor de Movimiento

Los sensores de movimiento nos permiten detectar cuando un ser humano se ha movido dentro o fuera de nuestro rango de sensado. Suelen ser pequeños, dependiendo de su alcance, y tienen gran aplicación en los hogares como base de sistemas de seguridad, pero también en los negocios y oficinas.

Se los conoce también como sensores infrarrojos pasivos, sensores infrarrojos o sensores piroeléctricos.

Estos sensores están hechos básicamente con materiales piroeléctricos, es decir, que tienen la capacidad de detectar radiación infrarroja. Todos los cuerpos que tienen temperatura emiten energía calórica en forma de radiación que es invisible al ojo humano, ya que se transmite a longitudes de onda infrarrojas. Luego estos sensores no miden dicha radiación, sino que son pasivos y son capaces de detectar dicha radiación para convertirla en un impulso eléctrico. Además suelen utilizar lentes Fresnel que permiten dirigir todas las ondas detectadas en un lugar único donde está el material piroeléctrico.

Para esta aplicación utilizamos un sensor piroeléctrico de movimiento HC-SR501 (ver Figura 6). Estos son sensores de movimiento de bajo costo cuya sensibilidad puede ser regulada. No nos permiten saber cuántas personas hay en el rango medible pero si la presencia o no de ellas.

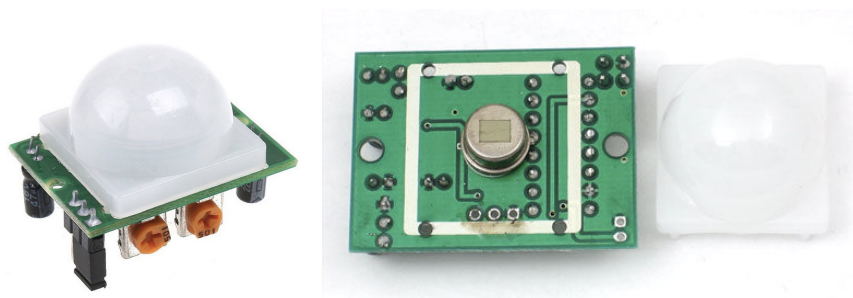


Figura 6. Sensor piroeléctrico de movimiento HC-SR501

Sensor de Luminosidad

Los sensores de luminosidad o sensores de luz son dispositivos electrónicos que responde al cambio de la intensidad de la luz. Estos sensores requieren de un componente emisor de luz, y un componente receptor que percibe dicha luz, permitiéndonos detectar el nivel de luz y producir una señal de salida representativa respecto a la cantidad de luz detectada.

Este tipo de sensores suelen ser aplicados en procesos industriales para optimizar el uso de la energía eléctrica, también en invernaderos para disminuir o aumentar la incidencia de luz solar. Además, es común encontrarlos en el uso cotidiano de sistemas de luz automáticos.

Estos sensores trabajan con un elemento transductivo o transductor, que son dispositivos eléctricos que permiten transformar la energía lumínica (fotones) en energía eléctrica. Existen sensores resistivos o transistivos. Los sensores resistivos trabajan con resistencias que varían su resistividad en base a la cantidad de luz recibida, generalmente llamadas fotoresistencias. Por otro lado, los sensores transistivos trabajan con fototransistores, que permiten el paso o no de corriente en base a la intensidad de luz recibida.

Para esta aplicación trabajamos con un módulo de sensor lumínico LDR BH1750FVI, que es un sensor lumínico de bajo costo incorporado a una placa que nos permite leer la intensidad de luz del ambiente de manera digital y analógica. Este módulo trabaja con una célula fotoeléctrica o fotocélula, que es un tipo de fotoresistencia.

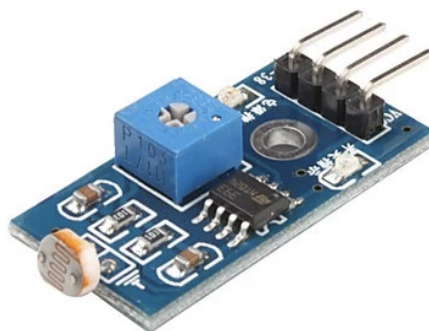


Figura 7. Módulo de Sensor lumínico LDR BH1750FVI

Sensor de Humedad del Suelo

Los sensores de humedad del suelo básicamente miden el volumen de agua en el suelo. La medición del suelo es importante para su aplicación agrícola, generalmente, para optimizar los sistemas de riego para que trabajen más eficientemente. No solo para utilizar la mínima cantidad de agua, sino también para lograr optimizar la calidad de las cosechas que dependen de estas características de manera crítica.

Este tipo de sensores permiten medir el volumen de agua mediante resistencias o capacitores. Los sensores resisitivos trabajan con resistencias que varían su resistividad dependiendo de su contacto con el agua, generalmente a mayor cantidad de agua la resistencia es menor ya que la electricidad corre más fácilmente, por lo que luego se puede determinar el volumen de agua en el suelo en base a dicha resistencia. También hay sensores capacitivos que trabajan de una manera similar a los sensores de humedad de ambiente capacitivos.

Para esta implementación hicimos uso de un módulo de sensor de humedad del suelo HL-69 (ver Figura 8). Básicamente consiste en dos placas separadas entre sí por una distancia determinada, que están cubiertas por un material conductor, cuya conductividad puede ser medible por un circuito de control. Este sensor nos permite medir la humedad del suelo a través de una salida digital o mediante una salida analógica.

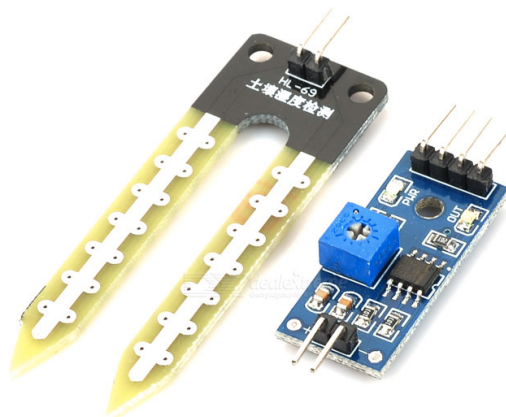


Figura 8. Módulo de Sensor de Humedad del Suelo HL-69

Actuadores

Los actuadores son dispositivos capaces de transformar energía hidráulica, neumática o eléctrica en la activación de un proceso con la finalidad de generar un efecto automatizado. Estos reciben una orden de un controlador y en función de ella activan o desactivan un elemento final de control, como por ejemplo, una válvula. Para nuestra implementación hicimos sólo uso de actuadores eléctricos que describiremos a continuación:

Relé

El relé o relevador es un dispositivo electromagnético que funciona como un interruptor controlado mediante un circuito eléctrico. Este circuito eléctrico consta de una bobina y un electroimán, que mediante el paso de corriente o la ausencia de ella, permite accionar uno o varios contactos para abrir o cerrar otros circuitos eléctricos independientes.

Los relés son actuadores binarios, es decir, que poseen dos estados posibles, abierto o cerrado. En cambio los motores son actuadores continuos, ya que pueden rotar 360 grados.

Principalmente los relés fueron utilizados en esta aplicación para controlar pasos de corriente, que consecuentemente, nos permitió el encendido y apagado de luces. Esto tiene aplicación en todo lo que es automatizado de luces, o de procesos que sean alimentados eléctricamente. Básicamente con la lectura de un sensor podemos decidir llevar a cabo una u otra acción.

Utilizamos módulos de relés de Arduino de distintos canales, los hay de 2, de 4, de 6, de 8 y más (ver Figura 9). Estos módulos nos permiten controlar con la salida de un pin GPIO, el paso de corriente de hasta 10 amperes con 250 voltios. Son de bajo costo y bastante confiables debido a la corriente y tensión que soportan.



Figura 9. Módulo de relés Arduino de 8 canales.

Electroválvulas

Una electroválvula es una válvula electromecánica, diseñada para controlar el paso de un fluido por un conducto, mediante el uso de un paso de corriente. Estas válvulas funcionan con una bobina solenoide ³, que convierte energía eléctrica, mediante magnetismo, en energía mecánica para abrir o cerrar la válvula. Generalmente no tienen más que dos posiciones, abiertas o cerradas.

Las electroválvulas se usan en muchas aplicaciones para controlar el flujo de todo tipo de fluidos. Suelen aplicarse a sistemas de riego, procesos industriales para realizar mezclas de fluidos, y otros. Además nos permiten medir el volumen del fluido en cuestión, ya que suelen venir un con diámetro preestablecido y podemos controlar la cantidad de tiempo que se encuentran abiertas o cerradas.

En este caso fueron utilizadas para controlar el paso de agua. Las electro válvulas que utilizamos son de 1/4 de pulgada (diámetro del conducto) y funcionan a 12 voltios, por lo que hicimos uso de los relés para poder controlar dicho voltaje (ver Figura 10).

³ Un solenoide es cualquier dispositivo físico capaz de crear un campo magnético sumamente uniforme e intenso en su interior y muy débil en el exterior. Generalmente una bobina de hilo conductor aislado y enrollado helicoidalmente.



Figura 10. Electroválvula de 1/4 de pulgada

HDMI CEC

HDMI (High Definition Multimedia Interface) o interfaz multimedia de alta definición, es una norma de audio y vídeo digital cifrado sin compresión. Este protocolo provee una interfaz entre cualquier fuente de audio y vídeo digital y un monitor de audio/vídeo digital compatible.

HDMI ofrece una función conocida como CEC (Consumer Electronics Control) o controles electrónicos de consumidores. CEC ofrece funciones como reproducción de un solo toque, el control del encendido y apagado y ajustes de configuraciones automatizadas. Este protocolo nos permite comunicarnos con hasta 15 dispositivos conectados en una red HDMI.

Si bien HDMI CEC no es un actuador propiamente dicho, se puede entender como tal, ya que los dispositivos Raspberry Pi poseen un puerto HDMI capaz de soportar CEC. Por lo tanto nos fue viable poder comunicarnos con una televisión a través de HDMI.

Aplicación

Para llevar a cabo una demostración de IntelliHome, se pensó en lograr una implementación para un hogar con tres ambientes, estos son, un living, una habitación y una zona exterior (ver Figura 11).

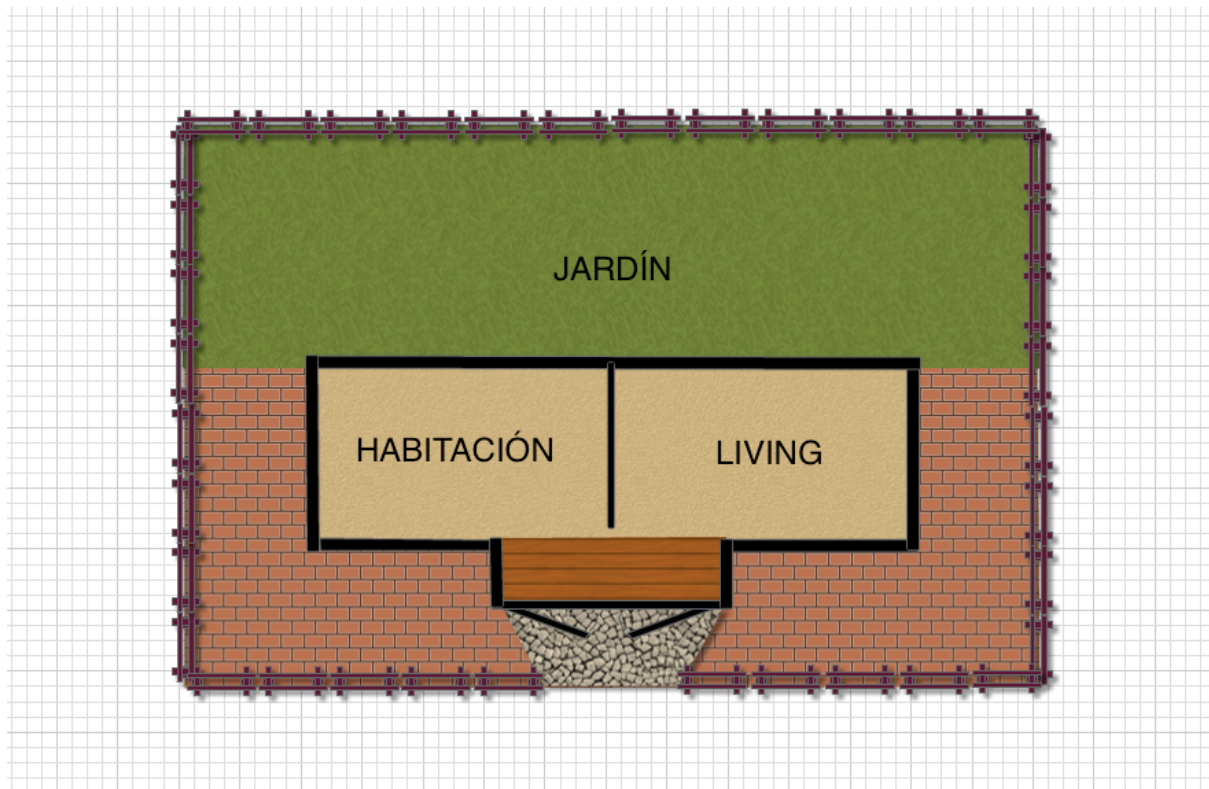


Figura 11. Plano del hogar con tres ambientes utilizado para la implementación

Teniendo en cuenta dicho plano realizamos algunas instalaciones haciendo uso de los dispositivos Raspberry Pi, los sensores y los actuadores descriptos. Utilizamos tres dispositivos Raspberry Pi, uno para cada ambiente del hogar.

En la habitación hicimos uso de un módulo de relés de 2 canales para controlar dos lámparas, y un sensor de humedad y temperatura DHT11. En el living, utilizamos otro módulo de relés de 2 canales para controlar dos lámparas, un sensor de humedad y temperatura, y además utilizamos HDMI CEC para controlar un televisor. Finalmente el jardín, utilizamos cuatro sensores de movimiento HC-SR501, un módulo de sensor lumínico LDR BH1750FVI, dos módulos de sensores de humedad del suelo HL-69, un módulo de relés de 8 canales para controlar 4 lámparas y una electroválvula de 1/4 de pulgada (ver Figura 12).

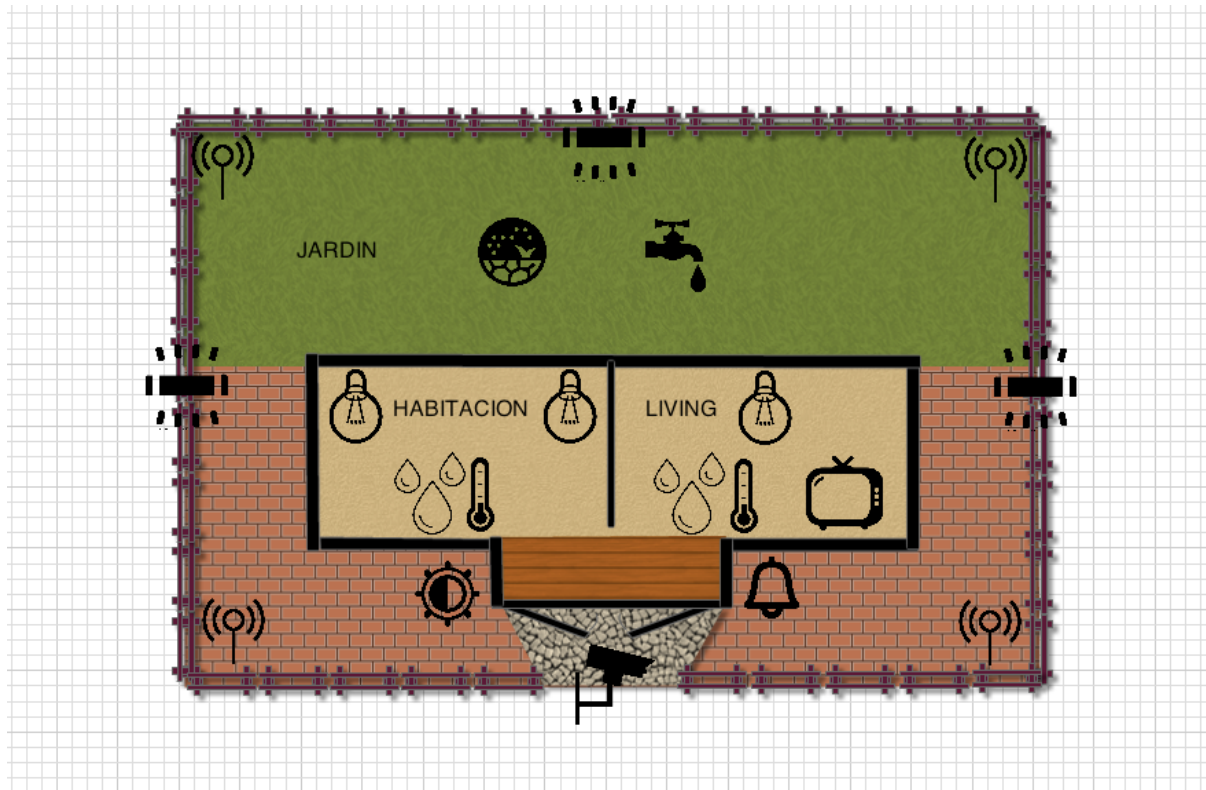


Figura 12. Plano del hogar indicando los componentes utilizados en la implementación.

A continuación se detallan los símbolos utilizados en el diagrama:



Sensor de humedad del ambiente



Sensor de temperatura del ambiente



Sensor de movimiento



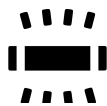
Sensor de humedad del suelo



Sensor de luminosidad



Lámparas interiores



Lámparas exteriores



Electroválvula



Televisión controlada por HDMI CEC



Cámara digital



Alarma

A continuación se muestra una imagen de la maqueta utilizada para demostrar la implementación (ver Figura 13).

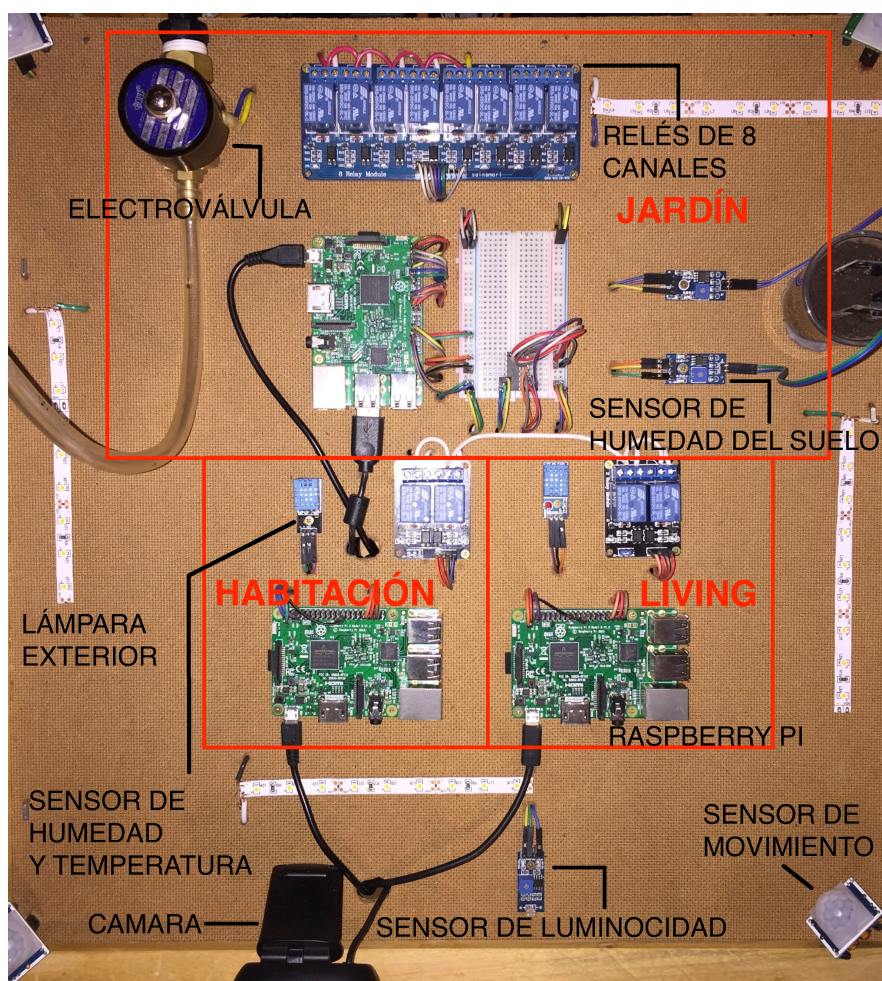


Figura 13. Plano del hogar indicando los componentes utilizados en la implementación.

integrado, colocado a la altura media de un protoboard ⁴. Los circuitos integrados son también conocidos como chips o microchips, y son estructuras pequeñas de material semiconductor sobre la que se fabrican circuitos electrónicos. En este caso se trata de un chip MCP3008, que actúa como conversor de señales analógicas en señales digitales. Para hacerlo, este chip lo que hace es muestrear hasta 200 mil veces por segundo la señal analógica, para que con la ayuda de una librería, podamos traducirlo a un valor numérico (ver Figura 15).

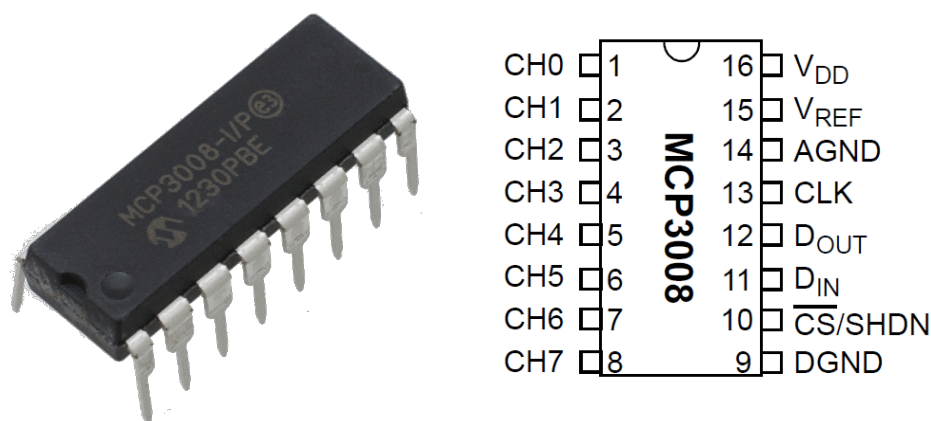


Figura 15. Microchip MCP3008 utilizado para la conversión de datos analógicos en digitales

Programas

Finalmente, para que todas estas conexiones tengan efecto son necesarios los programas correspondientes que se encargan de leer la información de los sensores mencionados, como también aquellos que se encargan de controlar los actuadores.

Se trata de un programa principal encargado de comunicarse con el servidor central, que inicia un programa por cada sensor que se desee leer. El programa principal recibe un archivo de configuración con los datos del servidor central, y con los datos de los sensores a leer, de forma tal que pueda saber qué programas iniciar para leer dichos sensores y cuál es la información necesaria para comunicarse con el servidor central a quien serán enviados estos datos recolectados.

⁴ Un protoboard o breadboard, es un tablero con orificios que se encuentran conectados eléctricamente entre sí de manera interna, habitualmente siguiendo patrones de líneas, en el cual se pueden insertar componentes electrónicos y cables para el armado y prototipado de circuitos electrónicos y sistemas similares.

Veamos un ejemplo de un archivo de configuración, a continuación se detalla el archivo de configuración utilizado para el dispositivo cliente Raspberry Pi ubicado en el ambiente exterior o jardín:

```
PING_SERVER_MULTICAST_IP = '224.1.1.1'
PING_SERVER_MULTICAST_PORT = 5007

NETWORK_DISCOVERY_SERVER_IP = '172.160.1.1'
NETWORK_DISCOVERY_SERVER_PORT = 5008

ROOM_ID = "4"

LIGHT_PINS = {"1":2, "2":3, "3":4, "4":17}
VALVE_PINS = {"1":27}
MOTION_SENSOR_PINS = {"1":6, "2":13, "3":19, "4":26 }
HUMIDITY_SENSOR_PINS = {}
TEMPERATURE_SENSOR_PINS = {}
LUMINOSITY_SENSOR_MCP3008_CHANNELS = {"1":7}
SOIL_HUMIDITY_SENSOR_MCP3008_CHANNELS = {"1":5, "2":6}
CAMERAS = {"1":"8081"}
```

Por un lado se detallan datos del servidor central como `PING_SERVER_MULTICAST_IP`, `PING_SERVER_MULTICAST_PORT`, `NETWORK_DISCOVERY_SERVER_IP` y `NETWORK_DISCOVERY_SERVER_PORT`, que denotan información para comunicarnos con el servidor. Luego también se indica el ID del dispositivo como `ROOM_ID = "4"`, y por último están denotados los pines a los cuales está conectado cada sensor. El programa principal que toma este archivo como entrada funciona en tres etapas de la siguiente manera:

El primer paso es conectarse a la red WiFi compartida con el servidor central. Luego el dispositivo Raspberry Pi se subscribe a una dirección multicast indicadas por las variables `PING_SERVER_MULTICAST_IP` y `PING_SERVER_MULTICAST_PORT`, en la cual el servidor central notificará cuál es su dirección IP dentro de la red doméstica WiFi.

Luego, una vez recibida esta información, el dispositivo Raspberry Pi analiza los sensores conectados a partir del archivo de configuración, e inicia un proceso independiente encargado de leer cada uno de esos sensores, para que comiencen a enviarle la información recolectada al servidor central. Veamos los distintos programas encargados de la lectura de cada sensor:

Publicador de sensor de luminosidad

A continuación se detalla la parte principal del programa encargado de leer el sensor de luminosidad y enviarlo a servidor central:

```
# Function to read SPI data from MCP3008 chip
def read_channel(channel):
    adc = spi.xfer2([1,(8+channel)<<4,0])
    data = ((adc[1]&3) << 8) + adc[2]
    return data

while True:
    time.sleep(5)
    previous_luminosity = current_luminosity
    current_luminosity = read_channel(sensor_mcp3008_channel)
    if current_luminosity != previous_luminosity:
        message = "rooms/" + room_id + "/luminosity_sensors/"
        message = message + sensor_id + "/" + str(current_luminosity)
        client.publish("luminosity_sensors", message)
```

Como se puede ver, este programa está realizando una lectura del sensor de luminosidad cada cinco segundos haciendo uso de la librería que nos permite controlar el chip MCP3008. En caso de que el estado actual del sensor sea distinto al estado anterior entonces se comunica el nivel de luminosidad al servidor central.

Publicador de sensor de movimiento

A continuación se detalla la parte principal del programa encargado de leer el sensor de movimiento y enviarlo a servidor central:

```
while True:
    previous_state = current_state
    current_state = GPIO.input(sensor_pin)
    if current_state != previous_state:
        if current_state == True:
            message = "rooms/" + room_id + "/motion_sensors/" + sensor_id + "/ON"
            client.publish("motion_sensors", message)
        else:
            message = "rooms/" + room_id + "/motion_sensors/" + sensor_id + "/OFF"
            client.publish("motion_sensors", message)
```

Como se puede ver en el programa indicado, estamos leyendo el sensor de movimiento continuamente a través de los pines GPIO del dispositivo Raspberry Pi. Cuando encontramos que el estado actual del sensor sufrió un cambio, entonces le comunicamos al servidor central el estado del mismo, ya sea activo o inactivo.

Publicador de sensor de humedad y temperatura del ambiente

A continuación se detalla la parte principal del programa encargado de leer el sensor de temperatura y humedad del ambiente, que luego envía dicha información al servidor central:

```
while True:
    time.sleep(5)
    previous_temperature = current_temperature
    previous_humidity = current_humidity
    current_humidity, current_temperature = DHT.read_retry(sensor_pin,
                                                            sensor_pin)

    if current_temperature != previous_temperature:
        if current_temperature is not None:
            temp_str = str('{0:0.1f}'.format(current_temperature))
            message = "rooms/" + room_id + "/temperature_sensors/"
            message = message + sensor_id + "/" + temp_str
            client.publish("temperature_sensors", message)
        else:
            message = "rooms/" + room_id + "/temperature_sensors/"
            message = message + sensor_id + "/failed"
            client.publish("temperature_sensors", message)

    if current_humidity != previous_humidity:
        if current_humidity is not None:
            humidity_str = str("{0:0.1f}".format(current_humidity))
            message = "rooms/" + room_id + "/humidity_sensors/"
            message = message + sensor_id + "/" + humidity_str
            client.publish("humidity_sensors", message)
        else:
            message = "rooms/" + room_id + "/humidity_sensors/"
            message = message + sensor_id + "/failed"
            client.publish("humidity_sensors", message)
```

Como se puede ver, este programa está realizando una lectura del sensor de humedad y temperatura del ambiente cada cinco segundos haciendo uso de la librería que nos provee DHT. En caso de que el estado actual tanto de la temperatura como de la humedad del sensor sea distinto al estado anterior de ambos respectivamente, entonces se comunica la temperatura en grados celcius y la humedad en porcentaje al servidor central.

Publicador de sensor de humedad del suelo

A continuación se detalla la parte principal del programa encargado de leer el sensor de humedad del suelo, que luego envía dicha información al servidor central:

```
# Function to read SPI data from MCP3008 chip
def read_channel(channel):
    adc = spi.xfer2([1, (8+channel)<<4, 0])
    data = ((adc[1]&3) << 8) + adc[2]
    return data

# Function to convert data to voltage level, rounded decimal places.
def convert_volts(data, places):
    volts = (data * 3.3) / float(1023)
    volts = round(volts, places)
    return volts

# Function to calculate humidity, rounded decimal places.
def convert_humidity(volts, places):
    humidity = (volts - 3.3) / float(-2.3 / float(950))
    humidity = round(humidity, places)
    return humidity

while True:
    time.sleep(2)
    previous_humidity = current_humidity
    channel_input = read_channel(sensor_mcp3008_channel)
    volts = convert_volts(channel_input, 2)
    current_humidity = convert_humidity(volts, 2)
    if current_humidity != previous_humidity:
        message = "rooms/" + room_id + "/soil_humidity_sensors/"
        message = message + sensor_id + "/" + str(current_humidity)
        print(message)
        client.publish("soil_humidity_sensors", message)
```

Como se puede ver, este programa está realizando una lectura del sensor de humedad del suelo cada dos segundos haciendo uso de la librería que nos permite controlar el chip MCP3008. En caso de que el estado actual del sensor sea distinto al estado anterior entonces se comunica el nivel de humedad del suelo al servidor central. Se puede notar que utilizamos una fórmula para convertir el voltaje que nos indica el sensor de humedad del suelo en un valor que nos resulte de interés.

Finalmente, la última etapa del dispositivo cliente Raspberry Pi es donde participan los actuadores. Como mencionamos, estos dispositivos no tienen inteligencia propia para decir que hacer con los actuadores, por lo que se comunican con el servidor central quien les dirá que hacer al respecto. Veamos los programas que nos permiten controlar los actuadores:

Actuador que controla luces a través de relés

A continuación se detalla la parte principal del programa encargado de controlar el actuador que controla las luces:

```
if action == 'ON':  
    print("Switching ON light (" + id + ")")  
    GPIO.output(LIGHT_PINS[id], GPIO.LOW)  
    client.publish("lights", "rooms/" + ROOM_ID + "/lights/" + id + "/ON")  
if action == 'OFF':  
    print("Switching OFF light (" + id + ")")  
    GPIO.output(LIGHT_PINS[id], GPIO.HIGH)  
    client.publish("lights", "rooms/" + ROOM_ID + "/lights/" + id + "/OFF")
```

Notar que estamos utilizando las salidas de los pines GPIO para controlar los relés que finalmente tienen las luces conectadas. Además una vez realizada la acción se vuelve a notificar al servidor central el estado final de las luces.

Actuador que controla electroválvulas a través de relés

A continuación se detalla la parte principal del programa encargado de controlar el actuador que controla las electroválvulas:

```
if action == 'ON':  
    print("Opening valve (" + id + ")")  
    GPIO.output(VALUE_PINS[id], GPIO.LOW)  
    client.publish("valves", "rooms/" + ROOM_ID + "/valves/" + id + "/ON")  
if action == 'OFF':  
    print("Closing valve (" + id + ")")  
    GPIO.output(VALUE_PINS[id], GPIO.HIGH)  
    client.publish("valves", "rooms/" + ROOM_ID + "/valves/" + id + "/OFF")
```

Notar que estamos utilizando las salidas de los pines GPIO para controlar los relés que finalmente tienen las electroválvulas conectadas. Además una vez realizada la acción se vuelve a notificar al servidor central el estado final de las electroválvulas.

Actuador que controla televisores a través de HDMI CEC

A continuación se detalla la parte principal del programa encargado de controlar el actuador que controla los televisores:

```
if action == 'ON':  
    print("Turning ON tv (" + id + ")")  
    os.system('echo "on "' + id + '" | cec-client -s')  
    client.publish("tvs", "rooms/" + ROOM_ID + "/tvs/" + id + "/ON")  
if action == 'OFF':  
    print("Turning OFF tv (" + id + ")")  
    os.system('echo "standby "' + id + '" | cec-client -s')  
    client.publish("tvs", "rooms/" + ROOM_ID + "/tvs/" + id + "/OFF")
```

Notar que estamos utilizando el protocolo HDMI CEC para controlar los televisores conectados a la red HDMI del dispositivo cliente Raspberry Pi. Además una vez realizada la acción se vuelve a notificar al servidor central el estado final de los televisores.

Configuración

Por último es importante comentar que es necesario un setup particular para los dispositivos Raspberry Pi. Es necesaria la instalación de ciertas librerías y otros elementos para lograr su funcionamiento en conjunto con el servidor central, y que los programas anteriormente descriptos puedan ejecutarse sin inconvenientes. Pasamos a detallar los pasos realizados en esta implementación que fueron necesarios para el setup de nuestros dispositivos Raspberry Pi.

Configurar el idioma de la Raspberry Pi

- Ejecutar `sudo dpkg-reconfigure locales` en una consola
- Deseleccionar la opción `en_GB.UTF-8 UTF-8`
- Seleccionar la opción `en_US.UTF-8 UTF-8`
- Confirmar la ventana presionando `OK`
- Confirmar el idioma `en_US.UTF-8 UTF-8`
- Copiar las siguientes líneas en el archivo `/etc/default/locale`

```
# File generated by update-locale  
LANG=en_US.UTF-8  
LC_CTYPE="UTF-8"  
LC_ALL="en_US.UTF-8"  
LANGUAGE="en_US.UTF-8"
```

Instalar ZSH

ZSH es un intérprete muy potente de comandos de shell, nos brinda muchas funcionalidades útiles que nos brindan mayor flexibilidad y simplicidad al momento de controlar nuestros dispositivos de manera remota. En una terminal ejecutar los siguientes comandos:

```
sudo apt-get install zsh
git clone git://github.com/robbyrussell/oh-my-zsh.git ~/.oh-my-zsh
cp ~/.zshrc ~/.zshrc.orig
cp ~/.oh-my-zsh/templates/zshrc.zsh-template ~/.zshrc
chsh -s /bin/zsh
```

Finalmente cerrar la terminal y abrir una nueva.

Clonar el repositorio de Github

En Github se encuentran publicados los programas necesarios para la lectura de sensores y controlado de actuadores que se describieron anteriormente. Primero crearemos una clave pública para agregarla al repositorio de Github y luego lo clonaremos, para ello es necesario ejecutar los siguientes comandos en una terminal:

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
cat ~/.ssh/id_rsa.pub          #Add the public key to your github account
git clone git@github.com:facuspagnuolo/domotic-client.git
```

Instalar un cliente CEC

El cliente CEC es necesario para controlar los televisores conectados a las Raspberry Pi por HDMI. Para ello necesitamos ejecutar los siguientes comandos en una terminal:

```
sudo apt-get update
sudo apt-get install cmake liblockdev1-dev
sudo apt-get install libudev-dev libxrandr-dev python-dev swig
cd /home
sudo git clone https://github.com/Pulse-Eight/platform.git
sudo mkdir platform/build
sudo git clone https://github.com/Pulse-Eight/platform.git
sudo mkdir platform/build
cd platform/build
```



```
sudo cmake ..
sudo make
sudo make install
cd /home
sudo git clone https://github.com/Pulse-Eight/libcec.git
sudo mkdir libcec/build
cd libcec/build
sudo cmake -DRPI_INCLUDE_DIR=/opt/vc/include -DRPI_LIB_DIR=/opt/vc/lib ..
sudo make -j4
sudo make install
sudo ldconfig
```

Finalmente podemos ejecutar los siguientes comandos para probar el encendido y apagado de nuestro televisor conectado al puerto HDMI del dispositivo Raspberry Pi:

```
echo "on 0" | cec-client -s      # Turn on the TV
echo "standby 0" | cec-client -s  # Turn off the TV
```

Instalar Motion

Motion es un servidor de streaming de video por cámaras. Este programa nos permite publicar las cámaras conectadas al dispositivo Raspberry Pi a través de un puerto que puede ser accedido por medio de Internet. Para ello realizamos los siguientes pasos:

- Instalamos el siguiente paquete de Motion `sudo apt-get install motion`
- Remplazamos el archivo de configuración por defecto de Motion `sudo cp Desktop/domotic-client/config/motion.conf /etc/motion/motion.conf`
- Configuramos Motion como proceso daemon editando el archivo `/etc/default/motion` y reemplazando `start_motion_daemon = no` por `yes`

Configuramos un cronetab

Configuraremos el programa principal del dispositivo Raspberry Pi de IntelliHome para que se ejecute al iniciar. Para ello editamos el archivo de crontabs con el comando `sudo crontab -e` y agregamos las siguientes líneas:

```
* * * * /etc/init.d/motion restart
@reboot sh /home/pi/Desktop/domotic-client/launcher.sh > /home/pi/Desktop/domotic-client/logs/cronlog 1>&1
```

Finalmente ejecutamos el siguiente comando para asegurarnos que el programa inicial sea ejecutable por nuestro sistema operativo: `sudo chmod +x launcher.sh`

Configurar SPI

SPI es la interfaz que utiliza la Raspberry Pi para comunicarse con el chip MCP3008 para leer datos analógicos, para poder hacer uso de esta debemos habilitarla editando el archivo `/etc/modules/` y agregando las siguientes líneas:

```
spidev  
bcm2708-rng
```

Instalar Adafruit DHT:

Adafruit DHT es una librería que nos permite leer los sensores de temperatura y humedad del ambiente DHT11. Existe una versión en Python que es la que utilizamos para esta implementación. Para instalarla debemos ejecutar las siguientes líneas:

```
cd /home  
sudo git clone https://github.com/adafruit/Adafruit_Python_DHT.git  
cd Adafruit_Python_DHT  
sudo python setup.py install
```

Instalar Mosquitto

Mosquitto es una implementación del protocolo MQTT⁵ que utilizamos en los dispositivos Raspberry Pi para enviar y recibir información del servidor central. Para ellos debemos ejecutar las siguientes líneas:

```
sudo apt-get install mosquitto  
cd /home  
sudo git clone https://github.com/eclipse/paho.mqtt.python.git  
cd paho.mqtt.python  
sudo python setup.py install
```

⁵ Ver página 37 para más información sobre este protocolo.

Servidor

Introducción

El servidor central fue implementado en Ruby y cumple varias funciones. Una de las más principales es la de recolectar los datos que le envían los clientes acerca de los sensores, por medio de los cuales puede conocer el estado del hogar. Además es el encargado de analizar el estado del hogar, es decir, la información recibida acerca de los sensores, para luego tomar decisiones que serán comunicadas nuevamente a los dispositivos clientes para llevar a cabo ciertas acciones.

Otra de las funciones principales del servidor central es la comunicación con el mundo exterior a través de internet. El servidor central expone una interfaz que nos permite interactuar con él. Dicha interfaz nos abstrae de la compleja arquitectura permitiéndonos configurarla y controlarla.

Además de permitirnos comunicarnos con la arquitectura, el servidor es el encargado de llevar a cabo la autenticación de usuarios. El servidor consta con una lista de usuarios registrados a los cuales se les permite gestionar el sistema. Es por eso que mediante credenciales compuestas por una dirección de correo electrónico y una contraseña de 16 caracteres, el servidor debe autenticar a esos usuarios de forma tal que se mantenga la seguridad de que sólo aquellos usuarios autorizados son los que gestionan la aplicación.

Por último el servidor central está comunicado con una base de datos PostgreSQL. En esta base de datos en donde están almacenados los usuarios, sus credenciales, el estado de los sensores y del resto del sistema, y las configuraciones que la aplicación requiere para funcionar correctamente.

Ruby

Es un lenguaje de programación interpretado, al igual que Python, reflexivo y orientado a objetos, creado por el programador japonés Yukihiro Matsumoto en 1993. Tiene una sintaxis inspirada en Python y Perl y características de programación orientada a objetos similares a Smalltalk. Su implementación oficial es distribuida bajo una licencia de software libre.

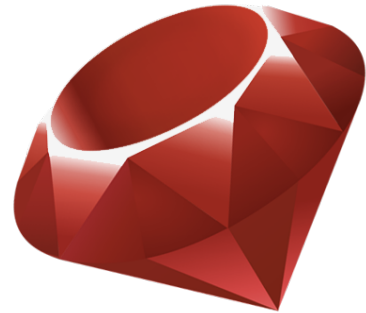


Figura 16. Logo del lenguaje Ruby

Es un lenguaje de programación reflexivo ya que posee la capacidad acceder a la representación dinámica de si mismo, esto es a la ejecución operacional del programa y de la implementación del lenguaje de programación. Existen lenguajes de programación que permiten reflexión estática o en tiempo de compilación. Otros casos de ejemplo, quizás los más comunes, son algunos lenguajes de programación de alto nivel que se ejecutan sobre máquinas virtuales, como Smalltalk o Java.

Yukihiro Matsumoto empezó a trabajar en Ruby en febrero de 1993, pero fue publicado recién en el año 1995. Su nombre deriva de una broma aludiendo al lenguaje de programación Perl. Ruby surge apuntando a la productividad y diversión del desarrollador, siguiendo los principios de una buena interfaz de usuario. Yukihiro sostenía que el diseño de sistemas necesitaba hacer hincapié en las necesidades humanas más que en las de la máquina, él decía que los ingenieros se centraban mucho en las máquinas pensando: *"Haciendo esto, la máquina funcionará más rápido. Haciendo esto, la máquina funcionará de manera más eficiente. Haciendo esto..."*

Por lo tanto, Ruby surge para centrarse en las personas, más aún, en cómo las personas hacen programas o cómo ellas manejan las aplicaciones en las computadoras.

Yukihiro explica que Ruby sigue el "*principio de la menor sorpresa*", lo que significa que el lenguaje debe comportarse de tal manera que minimice la confusión de los usuarios que lo utilizan, aunque luego trató de distanciarse de dicha filosofía. Finalmente aclaró que él usa un estándar personal de evaluación de la sorpresa que lo explicó de la siguiente manera: Todo el mundo tiene un pasado personal, puedes venir de Python, Perl, u otro, y aún así sorprenderse por los aspectos del lenguaje, por lo que dirían que Ruby viola el principio de la menor sorpresa. Por lo tanto debemos pensar en un principio que sea válido para todos, y eso significa un principio de la menor sorpresa después de haber aprendido bien Ruby. Incluso da el ejemplo de que fue programador de C++ durante tres años y que aún así se seguía sorprendiendo.

Tal como mencionamos, Ruby es un lenguaje de programación orientado a objetos en el que todos los tipos de datos son un objeto, incluidas las clases y también lo que otros lenguajes definen como primitivas como los tipos de datos, true, false y nil. En Ruby toda función es un método y las variables siempre referencian a un objeto y no los objetos mismos.

Ruby también soporta metaprogramación, es decir, que podemos escribir código que escribe código, o mejor dicho, código que escribe o manipula otros programas o a sí mismos como datos. También nos permite trabajar con full closures, que son objetos que representan un conjunto de colaboraciones, como los métodos, utilizados para parametrizar comportamiento y además nos permiten referencias al contexto en donde fueron creados.

Para la implementación del servidor central hicimos uso de la versión 2.2.1 de Ruby. Esta versión fue liberada en marzo del año 2015. Actualmente, ya se ha liberado la versión preview 2.4.0 de Ruby.

Paradigma OO

Cuando hablamos del paradigma orientado a objetos, es necesario mirar bien atrás para entenderlo en profundidad.

Cuando queremos resolver un problema, en este caso un problema de domótica, realizamos un modelo. Pero no cualquier modelo, sino un modelo computable, es decir, representamos un recorte de la realidad, específicamente esa tajada de cosas por medio de las cuales representamos el dominio del problema en cuestión, que a su vez puede ser ejecutado para resolverlo.

Luego, es importante modelar bien, pero qué quiere decir esto? Nos referimos a los tres ejes que tiene un modelo. Por un lado el eje funcional, es decir, qué tan buena es la representación del dominio del problema en nuestro modelo, de forma tal que si apareciese algo nuevo en el dominio, también debería aparecer en el modelo. Por otro lado el eje descriptivo, es decir, qué tan bien está descrito el modelo, si lo podemos comprender y por lo tanto cambiar, si utilizamos buenos nombres. Por último el eje implementativo, es decir, si podemos ejecutarlo en el tiempo esperado, si utiliza únicamente los recursos necesarios, también su escalabilidad y performance.

Por otro lado, qué es el desarrollo de software? Algunos pueden estar en desacuerdo, pero personalmente me gusta pensar en el desarrollo de software como un proceso de aprendizaje, iterativo e incremental. Si bien puede darse a la discusión, no podemos negar que siempre que debemos resolver un problema, no nacemos sabiendo cómo hacerlo. Primero, siempre analizamos el problema, lo entendemos, diseñamos una primer solución y por último la plasmamos en código. Pero luego, esto no estará terminado, aparecerán nuevas cosas por aprender, que deberemos volver a analizar, para volver a diseñar y finalmente volverlas a plasmar en nuestro código, una y otra vez.

Esto último comulga con el cambio como característica esencial del software, porque el dominio del problema puede cambiar, y en consecuencia cambiará nuestro entendimiento sobre dicho dominio. Por lo tanto, nuestros modelos se verán finalmente afectados. Lo cual contradice la visión clásica del desarrollo de software que trabajaba según el método cascada, en el cual cada paso se realizaba una única vez.

El paradigma orientado a objetos surge con Simula67, un lenguaje OO del año 1967 en el Centro de Cómputo Noruego en Oslo. Allí trabajan con simuladores de naves, y por lo tanto tuvieron la idea de agrupar diversos tipos de naves en diversas clases de objetos, siendo responsable cada clase de objetos de definir sus propios datos y comportamientos. Más tarde el paradigma orientado a objetos fue refinado en SmallTalk, otro lenguaje OO diseñado por Alan Kay.

Alan Kay propone un paradigma puro, en el cual todo es un objeto, y define un programa como un conjunto de objetos que interactúan entre sí enviándose mensajes. También define los objetos como una representación de un ente del dominio del problema, cuya esencia está modelada por los mensajes que comprende, que también son objetos. Finalmente, este paradigma nos aporta el concepto de clases, que son objetos que nos permiten representar ideas o conceptos, que también existen en la realidad.

Todos estos elementos que mencionamos del paradigma, nos ayudan a poder desarrollar software de la mano con nuestra manera de aprender. Por eso es que elijo este paradigma para resolver el problema de domótica en cuestión, ya que necesité aprender sobre el problema para luego plasmarlo en código, necesité cambiarlo porque aprendí cosas nuevas, y de mejor manera.

TDD

TDD significa Test Driven Development, es decir, desarrollo guiado por tests. Es una técnica que nos ayuda en el desarrollo de software para llevarla a cabo de manera iterativa e incremental. Es constructivista, ya que nos induce a ir construyendo aquellos modelos por medio de los cuales iremos resolviendo el problema en cuestión, a medida que vamos conociéndolo en profundidad.

TDD supone una serie de pasos para aplicar la técnica correctamente (ver Figura 17)

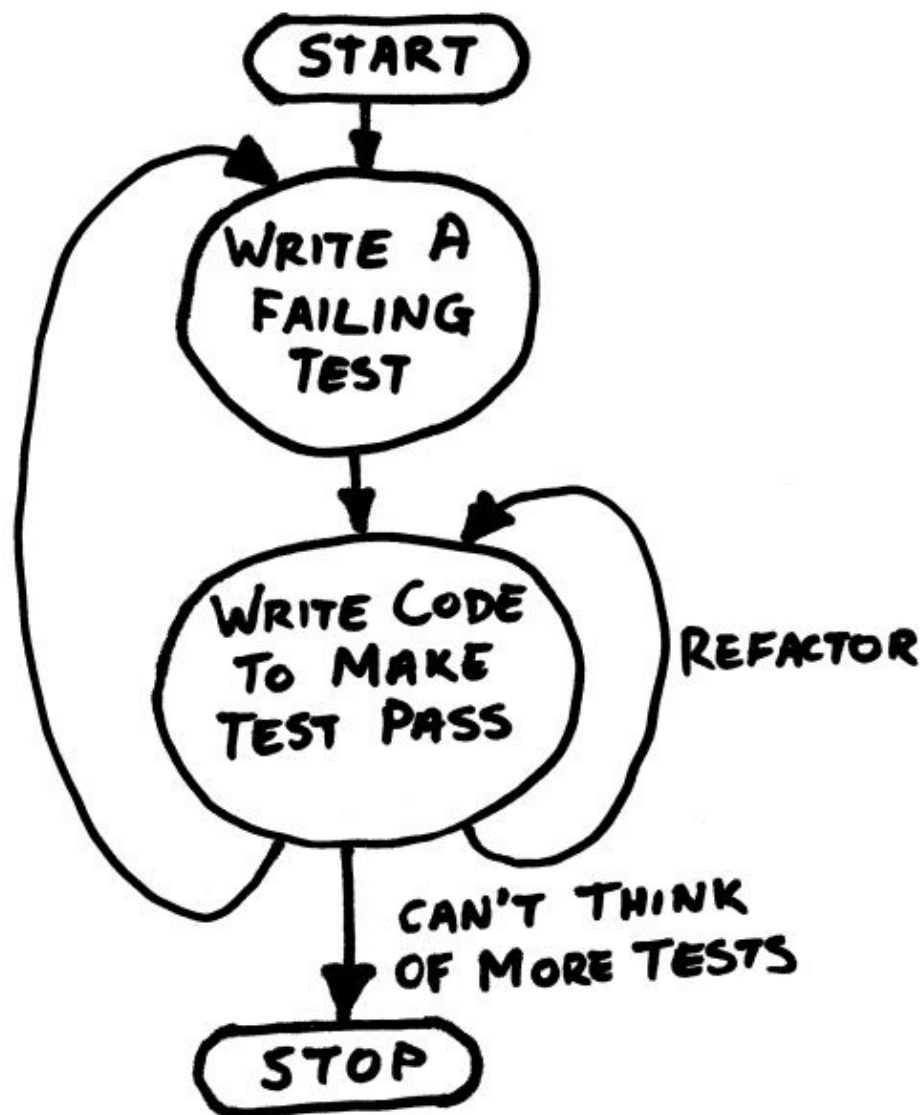


Figura 17. Etapas del ciclo de TDD

- 1- El primer paso es escribir un test, pero no cualquier test. Este test debe fallar, y además debe ser el más sencillo que se nos ocurra.
- 2- Implementar la solución más simple que haga pasar el primer test. Luego debemos correr todos los tests y asegurarnos que todos pasen, de lo contrario iteraremos hasta que esto suceda.
- 3- Reflexionamos si es posible o no mejorar el código. De ser cierto entonces refactorizaremos la solución, de lo contrario volveremos al primer paso.

Los pasos descriptos anteriormente tiene una explicación fundamental, basada en el proceso de aprendizaje. Debemos escribir un test que falle porque sino estaremos hablando de algo que ya conocemos. Además, debe ser el más sencillo, por que la idea está en buscar feedback inmediato, es decir, poder encontrar rápidamente algo nuevo por conocer y terminar de conocerlo en el menor tiempo posible. Una vez que aprendimos algo nuevo, es decir, que logramos que nuestro nuevo test pase, entonces ningún otro test debe estar en rojo, porque TDD recuerda todo lo aprendido y nos ayuda a no olvidarnos que lo que ya aprendimos. Finalmente, tenemos la posibilidad de emprolijar la solución confiándonos en que si algo de lo que hacemos no se condice con lo aprendido, TDD nos lo recordará.

Debemos notar que TDD tiene varias ventajas. Una de sus principales es lo que se conoce como YAGNI o *You Ain't Gonna Need It*, que básicamente se refiere que el programador no debería escribir código innecesario, funcionalidad no requerida. Siempre intentamos escribir el menor código posible que nos permite resolver el problema que planteamos. Otra ventaja es que cada vez que realizamos TDD, agregar más conocimiento a nuestro sistema se vuelve seguro, porque estaremos confiados que nada de lo que hemos hecho hasta ese entonces dejará de andar. Por lo tanto, podemos hacer grandes modificaciones o refactors sin miedo.

Otra gran ventaja es que TDD acompaña el proceso de aprendizaje del ser humano, cada cosa nueva que aprendemos se verá reflejada en un nuevo test, un valor agregado al sistema. Por lo tanto, también podemos pensar los tests como una documentación de todo lo aprendido, de todo lo que el sistema hace, luego no es

necesario actualizar ninguna documentación extra, los tests estarán siempre actualizados.

También debemos reconocer que TDD supone que no necesitamos aprender de la herramienta con la que estamos trabajando, ya sea el IDE, el lenguaje de programación, un framework, entre otros. TDD supone que nos concentramos pura y exclusivamente en el dominio del problema que queremos resolver.

Es muy común perderse debido a otros factores de forma tal que terminemos no haciendo TDD. Esto sucede cuando tardamos mucho tiempo en escribir un test que falle, o tardamos mucho tiempo en hacer pasar un test nuevo. También puede ser cuando escribimos tests que pasan directamente, o soluciones grandes que hacen pasar un test.

Por último, es importante entender que TDD no es hacer testing. TDD es técnica de aprendizaje que en consecuencia nos hace testear nuestro sistema. En cambio, podríamos programar muchas funcionalidades y luego escribir algunos tests para asegurarnos que dicha funcionalidad se comporta como tal, pero no habremos hecho TDD.

Para el desarrollo del backend de IntelliHome se aplicó TDD, no en su totalidad pero si en mayor parte. No se realizó lo mismo para el desarrollo de los dispositivos cliente ya que se trataba de una tecnología y un entorno que no se conocían. No conocíamos el lenguaje, ni las librerías y ni siquiera las propias Raspberry Pi. Para aplicar TDD en Ruby hicimos uso de la librería Rspec que fue desarrollada en 2005 por Steven Baker. Más adelante mostraremos algunos fragmentos de código desarrollado con TDD.

Rails

Rails también conocido como RoR o Ruby on Rails, es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby, que incluye todo lo necesario para crear aplicaciones web de acuerdo a la arquitectura MVC (Model View Controller) o Modelo Vista Controlador.



Figura 18. Logo de Ruby on Rails

Entender cómo es una arquitectura MVC es un paso clave para entender Rails. MVC se basa en dividir nuestras aplicaciones en tres capas, cada una de ellas con una responsabilidad diferente. Por un lado se encuentran las vistas, que Rails implementa a modo de templates que son responsables de proveer una representación de nuestra aplicación y sus recursos, lo que comúnmente llamamos interfaz. Los templates de Rails pueden ser en varios formatos, aunque mayormente se utilizan templates HTML embebidos con código Ruby.

Por otro lado están los modelos, que representan los modelos del dominio del problema que pretendemos resolver. Allí se encapsula toda la lógica del negocio específica a la aplicación. Finalmente existen los controladores, que pertenecen a la capa responsable de manejar los requests HTTP que llegan al servidor y gestionar una respuesta adecuada para cada uno de ellos. Estos son los encargados de interactuar con los modelos y las vistas para llevar a cabo la petición requerida en cada request.

Rails fue desarrollado por David Heinemeier Hansson, más conocido como DHH, a partir de su trabajo realizado en Basecamp, una herramienta de gestión de proyectos. Fue liberado al público por primera vez en julio de 2004. Rails surge con la idea de simplificar el desarrollo de aplicaciones web, con el objetivo de poder combinar la simplicidad al momento de escribir código para nuestra aplicación con la menor configuración posible, a diferencia de otros frameworks.

Rails se basa en dos principios. Por un lado el principio de DRY o *Don't repeat Yourself*, es decir, *no te repitas*. Este principio significa que las definiciones que hagamos en nuestra aplicación deberían hacerse sólo una vez. Las definiciones de clases, la manera en que están persistidos nuestros objetos, el comportamiento de los mismos, debe encontrarse en un sólo lugar y además Rails se encarga de deducir lo necesario para evitar que debamos repetirlo. Por otro lado, el principio de convención sobre configuración, que significa que el programador sólo debe encargarse de definir aquella configuración que sea no convencional. Es decir, Rails establece muchas convenciones por defecto, haciendo que todos los componentes de nuestra aplicación sigan esas convenciones, por lo tanto, si respetamos las convenciones estaremos en un mundo feliz. Casi no tendremos que configurar el framework, sólo lo haremos en caso que queramos indicar que no queremos seguir esas convenciones.

Quizás `ActiveRecord` sea el pilar más importante de Rails, esta es la parte de Rails que implementa su ORM ⁶. En Rails basta con que una clase herede de `ActiveRecord::Base` para que Rails interprete que esa clase será persistida en una table cuyo nombre será igual al plural del nombre de dicha clase. `ActiveRecord` nos da la ventaja de que casi no debemos ensuciar nuestras clases con configuraciones de persistencia. Más adelante mencionaremos la aplicación de algunas de estas convenciones en el desarrollo del backend de IntelliHome.

Rails actualmente se distribuye a través de RubyGems, que es el formato oficial de paquetes y canal de distribución de bibliotecas y aplicaciones Ruby.

Se tomó la decisión de utilizar Rails para desarrollar el servidor central de IntelliHome ya que verdaderamente Rails es un excelente framework para un kick-off. Es realmente fácil escribir los primeros casos de test, para luego implementar

⁶ ORM (Object Relational Mapping) o mapeo objeto-relacional es el conjunto de técnicas de programación que implementan muchos frameworks para poder persistir objetos del paradigma orientado a objetos en bases de datos relacionales.

nuestras ideas y finalmente poner en producción nuestra aplicación y probarla. Además RubyGems nos provee muchas librerías interesantes para resolver problemas comunes en el ámbito de las aplicaciones web, como la autenticación, el manejo de usuarios, o mismo los frameworks de tests.

Pero también debemos ser honestos. A Rails hay que tratarlo con cuidado, cuando las aplicaciones empiezan a crecer ya todo no es tan feliz. En Rails todo se caracteriza por ser rápido, y muchas veces nos enviamos de las convenciones de Rails para resolver problemas en poco tiempo a tal punto que no pensamos si es una buena solución, y ni si quiera si está bien aplicar las convenciones de Rails para ese caso concreto. Por lo tanto, es recomendable quizás tomarse algunos minutos más para pensar la solución y pensar en su escalabilidad al momento de resolverla con Rails.

Además, solemos siempre utilizar gemas ya escritas para resolver cualquier tipo de problemas. RubyGems está repleto de gemas mal implementadas, gemas que ensucian nuestro código y que, si bien resuelven el problema, nos traen otros nuevos que quizás no distingamos de entrada. Es por eso que también es recomendable pensar en escribir nuestras propias gemas, o pensar en varias alternativas antes de meter una gema nueva en nuestra aplicación para resolver un problema.

Es importante no perder de vista que estamos trabajando en el paradigma orientado a objetos. Esto es, recordar siempre que, para resolver el dominio del problema en cuestión, es importantísimo entender la esencia de los objetos que interactúan en ese dominio para poder representarlo en nuestro código.

Hace unos mese escribí un post sobre estas desventajas de Rails llamado *Ruby on Pains*. Allí se explican varios ejemplos y casos concretos que muestran lo que mencionamos antes ⁷.

⁷ Ver el post Ruby on Pains - blog.10pines.com/2015/11/25/ruby-on-pains/

PostgreSQL

PostgreSQL, también conocido como Postgres es un sistema de gestión de bases de datos relacional orientado a objetos. Es una base de datos open source y de uso gratuito. Nace a raíz de un proyecto llamado Ingres, en Berkeley en la década del 70 como un intento de competencia a las bases de datos de Oracle. Dicho proyecto continuó en 1985, luego de que su mentor Michael Stonebraker había dejado la universidad en el año 1982. Este nuevo comienzo lo

llamaron Postgres, y su nombre se debe a la analogía de post-Ingres. En 1989 recién se publicó la primera versión de esta base de datos. Como muchos otros proyectos de código abierto, el desarrollo de Postgres no es manejado por una empresa, sino que es dirigido por una comunidad de desarrolladores que trabajan de manera desinteresada, apoyados por organizaciones comerciales. Esta comunidad se la llamó PGDG o *PostgreSQL Global Development Group*.



Figura 19. Logo de PostgreSQL

Este tipo de bases de datos son una extensión de las bases de datos relacionales tradicionales, a las cuales se les proporcionan características de la programación orientada a objetos. Las bases de datos relacionales orientadas a objetos surgen debido al avance de los ORM, muchas aplicaciones escritas en lenguajes orientados a objetos siguen trabajando con bases de datos relacionales para llevar a cabo su persistencia.

Para la implementación no se requería una base de datos para hacer cosas complejas, esta iba a estar ubicada en el mismo hogar a controlar, junto con el servidor central, por lo que tampoco necesitamos manejar grandes cantidades de datos. Por lo tanto decidimos priorizar una fácil instalación y poca configuración.

Para el proyecto de IntelliHome se tuvieron en cuenta dos opciones, Postgres y MySQL. Finalmente se decidió utilizar Postgres debido a que se trata de un sistema de gestión de base de datos relacional orientado a objetos, a diferencia de MySQL que es simplemente relacional. Además, si bien Rails es agnóstico a la base de datos, por lo que existen una amplia variedad de opciones, Postgres sigue siendo una de las más recomendadas actualmente por la comunidad.

Implementación

El servidor central cumple varias funciones. Una de las más importantes es la de recolectar los datos que le envían los clientes acerca de los sensores, para conocer el estado del hogar, y luego analizarlo para luego tomar decisiones que serán comunicadas a los dispositivos para llevar a cabo ciertas acciones. También es el encargado del manejo de usuarios y su autenticación.

Modelo

La implementación de IntelliHome está soportada por un modelo que principalmente se encarga de representar los ambientes de un hogar, junto con los artefactos que se pueden controlar y los sensores encargados de captar información acerca del estado de los ambientes. El modelo diseñado se describe a continuación (ver Figura 20)

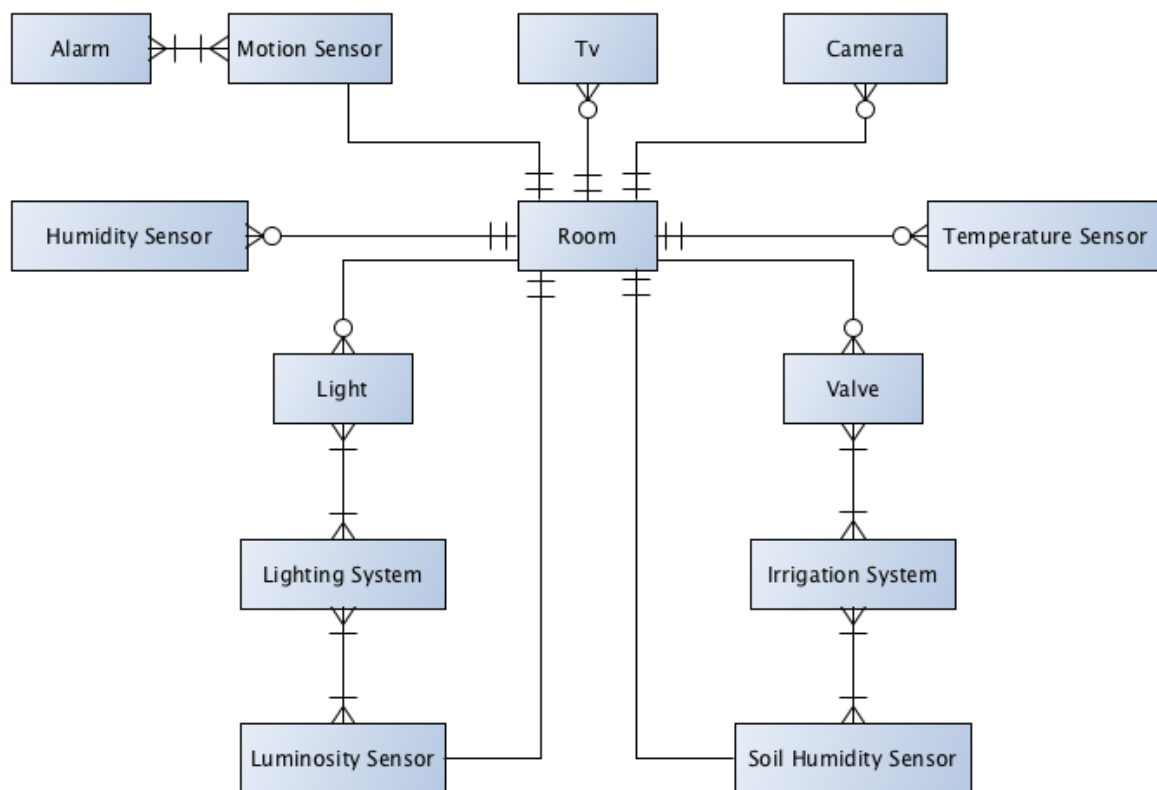


Figura 20. Diagrama de clases

Como se puede ver en el diagrama de clases, tenemos una entidad central que modela los ambientes de un hogar (room). Esta entidad tiene conocimiento de varias otras, entre ellas un ambiente puede contener ninguna o varias luces (light), que es la entidad que modela propiamente las luces de un hogar. Una luz debe pertenecer a un único ambiente, al igual que las válvulas (valve). Estas últimas modelan las canillas de agua o los pases de agua que puede haber en los ambientes de un hogar. Las válvulas están pensadas más para ambientes exteriores como un jardín, aunque también podrían utilizarse para las cocinas o baños.

Las habitaciones también pueden tener ninguna o varias televisiones (tv) así como ninguna o varias cámaras (camera). A su vez una televisión debe pertenecer sólo a un único ambiente al igual que cada cámara.

Por otro lado se encuentran los sensores que son cinco, los sensores de movimiento (motion sensor), los sensores de luminosidad (luminosity sensor), los sensores de humedad del suelo (soil humidity sensor), los sensores de humedad del ambiente (humidity sensor) y los sensores de temperatura (temperature sensor). Cada uno de ellos debe estar contenido en una única habitación o ambiente como podemos ver en el diagrama.

Las habitaciones no tienen conocimiento directo sobre los sensores de humedad del suelo, los sensores de movimiento y los sensores lumínicos, ya que se decidió modelar algunos sistemas encargados de encapsular cierta lógica más compleja sobre la interacción con estos sensores que será descripta más adelante. Estos sistemas son los sistemas de riego (irrigation system), los sistemas de iluminación (lighting system) y los sistemas de alarmas (alarm). Por el otro lado, las habitaciones sólo tienen conocimiento de los sensores de temperatura y de los sensores de humedad del ambiente, dado que para la implementación de IntelliHome no se han implementado sistemas particulares con el objetivo de gestionar estos sensores.

Sistemas

Tal como se mencionó recientemente, se decidió implementar algunos sistemas para lograr un control más óptimo y complejo de ciertos elementos de los ambientes de un hogar. Para esta implementación se modelaron tres tipos de sistemas, los sistemas de riego, los sistemas de iluminación, y los sistemas de alarmas.

Los sistemas implementados básicamente combinan los sensores con los actuadores. Cuando se reciben los datos recolectados por los distintos tipos de sensores, los distintos sistemas son notificados. Luego cada sistema implementa una lógica encargada de evaluar si se debe o no ejecutar una acción sobre los actuadores asociados. Veamos cada uno de ellos

Sistemas de Iluminación

Los sistemas de iluminación fueron implementados para el manejo de luces de manera automática. Básicamente combinan los sensores de luminosidad con los relés que son los actuadores encargados de manejar los artefactos lumínicos del hogar.

Los sensores de luminosidad reportan un valor que corresponde a la intensidad de luz captada en el ambiente. Cada vez que la intensidad de la luz captada por un sensor se modifica, el servidor central es notificado al respecto. Existen unos objetos particulares dentro del servidor encargados de estar atentos a esas notificaciones para luego comunicárselas a los sistemas de iluminación.

Cada sistema de iluminación posee un umbral de intensidad de luz. Este umbral es un valor numérico configurable, y principalmente participa en la actividad o inactividad del sistema con la siguiente lógica: si el sistema está inactivo, es decir, todos sus relés están desactivados, y los sensores de luminosidad asociados pasan el umbral, luego el sistema se activará y en consecuencia activará todos los relés asociados. Por otro lado, si el sistema está activo, es decir, al menos uno de sus relés está activado, y los sensores de luminosidad asociados no pasan el umbral, luego el sistema se desactivará y en consecuencia se desactivarán todos los relés.

Cabe destacar que este modelo soporta la inhabilitación o habilitación particular de cada sensor de luminosidad. Esto se utiliza para discriminar los sensores inhabilitados al momento de evaluar la lógica de activación o desactivación del sistema. Es decir, podemos ignorar un conjunto de sensores como si no formaran parte del sistema. Además los propios sistemas de iluminación pueden a su vez ser inhabilitados por completo, es decir, aunque alguna de las condiciones mencionadas anteriormente sobre la lógica de funcionamiento sea satisfecha, luego el sistema no se activará ni se desactivará el sistema.

A continuación se detallan los mensajes principales que responden los objetos que modelan los sistemas de iluminación:

```
class LightingSystem < ActiveRecord::Base
  has_and_belongs_to_many :lights
  has_and_belongs_to_many :luminosity_sensors

  def disabled?
    !enabled?
  end

  def enable!
    update_attributes(enabled: true)
  end

  def disable!
    update_attributes(enabled: false)
  end

  def active?
    lights.any?(&:switched_on?)
  end

  def inactive?
    !active?
  end

  def activate_or_inactivateDepending_on_sensors(on_activate_block, on_inactivate_block)
    if inactive? && do_all_the_enabled_sensors_reached_the_threshold?
      activate!
      on_activate_block.call
    elsif active? && do_not_all_the_enabled_sensors_reached_the_threshold?
      inactivate!
      on_inactivate_block.call
    end
  end

  def activate!
    lights.each do |light|
      mqtt_publisher(light, 'ON').call { return puts "Could not switch #{light.name} ON" }
      light.switch_on!
    end
  end

  def inactivate!
    lights.each do |light|
      mqtt_publisher(light, 'OFF').call { return puts "Could not switch #{light.name} OFF" }
      light.switch_off!
    end
  end

  private

  def do_all_the_enabled_sensors_reached_the_threshold?
    luminosity_sensors.enabled.all? { |sensor| sensor.passes? luminosity_threshold }
  end

  def do_not_all_the_enabled_sensors_reached_the_threshold?
    luminosity_sensors.enabled.all? { |sensor| !(sensor.passes? luminosity_threshold) }
  end
end
```

La lógica principal de activación o desactivación del sistema de iluminación se encuentra en el mensaje `activate_or_inactivateDepending_on_sensors`, tal como se describió anteriormente.

Sistemas de Riego

Los sistemas de riego fueron implementados para el manejo de válvulas de paso de agua de manera automática. Básicamente combinan los sensores de humedad del suelo con las electro-válvulas encargadas de manejar los pasos de agua del hogar.

Los sensores de humedad del suelo reportan un valor que corresponde a la humedad captada en la tierra. Cada vez que la humedad del suelo captada por un sensor se modifica, el servidor central es notificado al respecto. Existen unos objetos particulares dentro del servidor encargados de estar atentos a esas notificaciones para luego comunicárselas a los sistemas de riego.

Cada sistema de riego posee un umbral de humedad. Este umbral es un valor numérico configurable, y principalmente participa en la actividad o inactividad del sistema con la siguiente lógica: si el sistema está inactivo, es decir, todas sus electro-válvulas están cerradas, y los sensores de humedad del suelo asociados no pasan el umbral, luego el sistema se activará y en consecuencia se abrirán todas las electro-válvulas asociadas. Por otro lado, si el sistema está activo, es decir, al menos una de sus electro-válvulas está abierta, y los sensores de humedad del suelo asociados pasan el umbral, luego el sistema se desactivará y en consecuencia se cerrarán todas las electro-válvulas.

Cabe destacar que este modelo soporta la inhabilitación o habilitación particular de cada sensor de humedad del suelo. Esto se utiliza para discriminar los sensores inhabilitados al momento de evaluar la lógica de activación o desactivación del sistema. Es decir, podemos ignorar un conjunto de sensores como si no formaran parte del sistema. Además los propios sistemas de riego pueden a su vez ser inhabilitados por completo, es decir, aunque alguna de las condiciones mencionadas anteriormente sobre la lógica de funcionamiento sea satisfecha, luego el sistema no se activará ni se desactivará el sistema.

A continuación se detallan los mensajes principales que responden los objetos que modelan los sistemas de riego:

```
class IrrigationSystem < ActiveRecord::Base
  has_and_belongs_to_many :valves
  has_and_belongs_to_many :soil_humidity_sensors

  def disabled?
    !enabled?
  end

  def enable!
    update_attributes(enabled: true)
  end

  def disable!
    update_attributes(enabled: false)
  end

  def active?
    valves.any?(&:opened?)
  end

  def inactive?
    !active?
  end

  def activate_or_inactivateDepending_on_sensors(on_activate_block, on_inactivate_block)
    if inactive? && do_not_all_the_enabled_sensors_reached_the_threshold?
      activate!
      on_activate_block.call
    elsif active? && do_all_the_enabled_sensors_reached_the_threshold?
      inactivate!
      on_inactivate_block.call
    end
  end

  def activate!
    valves.each do |valve|
      mqtt_publisher(valve, 'ON').call { return puts "Could not open #{valve.name}" }
      valve.open!
    end
  end

  def inactivate!
    valves.each do |valve|
      mqtt_publisher(valve, 'OFF').call { return puts "Could not close #{valve.name}" }
      valve.close!
    end
  end

  private

  def do_not_all_the_enabled_sensors_reached_the_threshold?
    soil_humidity_sensors.enabled.all? { |sensor| sensor.is_below? humidity_threshold }
  end

  def do_all_the_enabled_sensors_reached_the_threshold?
    soil_humidity_sensors.enabled.all? { |sensor| !(sensor.is_below? humidity_threshold) }
  end
end
```

La lógica principal de activación o desactivación del sistema de riego se encuentra en el mensaje `activate_or_inactivateDepending_on_sensors`, tal como se describió anteriormente.

Sistemas de Alarma

Los sistemas de alarma fueron implementados para el manejo de alarmas sonoras de seguridad. Básicamente combinan los sensores de movimiento con una alarma sonora ubicada en el servidor central.

Los sensores de movimiento reportan si existe la presencia o no de movimiento de un cuerpo. Cada vez que alguno de estos sensores detecta la presencia de movimiento, el servidor central es notificado al respecto. Existen unos objetos particulares dentro del servidor encargados de estar atentos a esas notificaciones para luego comunicárselas a los sistemas de alarma.

A diferencia de los sistemas detallados anteriormente, las alarmas no poseen un umbral ni tampoco interesa persistir el estado de los sensores de movimiento ya que es un valor binario frente al cual se debe tomar una decisión inmediata. Basta con que uno de ellos detecte un movimiento, para que la alarma comience a sonar.

La desactivación de la alarma es manual, es decir, no basta con que todos los sensores no detecten más movimientos para que el sistema termine de sonar, sino que manualmente debemos desactivarlo.

Al igual que con los sistemas de riego y los de iluminación, podemos inhabilitar o habilitar particularmente cada sensor de movimiento asociado a una alarma. Esto se utiliza para discriminar los sensores inhabilitados al momento de activar la alarma. Es decir, podemos ignorar un conjunto de sensores como si no formaran parte de la misma. Además las propias alarmas pueden a su vez ser inhabilitadas por completo, es decir, aunque alguno de los sensores de movimiento detecte movimiento alguno, la alarma no sonará.

Se decidió modelar estos tres sistemas únicamente ya que son los más intuitivos y de uso cotidiano. Pero debemos notar que el potencial que posee este tipo de modelos es aún mayor. Simplemente pensemos en la idea de que en base a la lectura de muchos tipos de sensores, podemos interactuar con muchos otros tipos

de actuadores, lo que nos da una flexibilidad inmensa sobre la funcionalidad que podemos controlar en un hogar.

Eventos

Como mencionamos en la descripción de los sistemas de iluminación, los sistemas de riego y los sistemas de alarma, cada vez que alguno de los sensores que interactúan con estos sistemas sufre un cambio, el servidor central es notificado para luego intervenir en los sistemas correspondientes. También dijimos que existían unos responsables de escuchar estos cambios y notificarlo al resto de la aplicación. Estos objetos se los denominó *EventHandler*, es decir, controlador de eventos.

Existen distintos tipos de *EventHandler*, uno especializado para cada sensor sobre el cual se recibe información. Estos objetos son los que reciben la notificación de que hubo información nueva recolectada por un sensor, para luego interactúan con el resto del sistema.

Existen dos controladores de eventos para los sensores de humedad y temperatura del ambiente llamados `HumiditySensorEventHandler` y `TemperatureSensorEventHandler` respectivamente. Estos sólo están encargados de actualizar el valor de humedad y de temperatura de los objetos que modelan estos sensores en el servidor central.

Existe un controlador de eventos para los sensores de luminosidad llamado `LuminositySensorEventHandler` encargado de actualizar el valor de luminosidad de los objetos que modelan estos sensores en el servidor central, y de notificar a los sistemas de iluminación.

De la misma manera que el anterior, existe un controlador de eventos para los sensores de humedad del suelo llamado `SoilHumiditySensorEventHandler` encargado de actualizar el valor de humedad de los objetos que modelan estos

sensores en el servidor central, y de notificar a los sistemas de riego. Y por último existe un controlador de eventos para los sensores de movimiento llamado `MotionSensorEventHandler` encargado de activar a las alarmas.

Resultó muy interesante desarrollar estos controladores de eventos utilizando TDD, así que veamos como ejemplo el caso del `LuminositySensorEventHandler`:

```
require 'spec_helper'

describe EventHandler::SoilHumiditySensorEventHandler do
  let(:room) { FactoryGirl.create(:room) }
  let(:soil_humidity_sensor) {
    FactoryGirl.create(:soil_humidity_sensor, :enabled, room: room)
  }
  let(:event_handler) {
    EventHandler::DeviceEventHandler.for_soil_humidity_sensor
  }

  describe '#receive' do
    it 'updates the soil humidity sensor soil humidity value' do
      soil_humidity_sensor_value = 800

      event_handler.receive(soil_humidity_sensor, soil_humidity_sensor_value)
      sensors_humidity = soil_humidity_sensor.reload.soil_humidity

      expect(sensors_humidity).to eq soil_humidity_sensor_value
    end

    context 'when there is a lighting system depending on that sensor' do
      let(:humidity_threshold) { 900 }
      let!(:irrigation_system) {
        FactoryGirl.create(:irrigation_system, :enabled,
          soil_humidity_sensors: soil_humidity_sensors,
          humidity_threshold: humidity_threshold
        )
      }

      before do
        mqtt_publisher_mock
        event_handler.receive(soil_humidity_sensor, soil_humidity_sensor_value)
      end

      context 'when the soil humidity sensor is the only one' do
        let(:soil_humidity_sensors) { [soil_humidity_sensor] }

        context 'when the soil humidity pass the threshold' do
          let(:soil_humidity_sensor_value) { humidity_threshold + 1 }

          it 'inactivates the lighting system' do
            expect(irrigation_system.reload).to be_inactive
          end
        end

        context 'when the soil humidity is below the threshold' do
          let(:soil_humidity_sensor_value) { humidity_threshold - 1 }

          it 'activates the lighting system' do
            expect(irrigation_system.reload).to be_active
          end
        end
      end
    end
  end
end
```



```
context 'when there are more than one soil humidity sensor' do
  let(:soil_humidity_sensors) {
    [soil_humidity_sensor, another_soil_humidity_sensor]
  }
  let(:another_soil_humidity_sensor) {
    FactoryGirl.create(:soil_humidity_sensor,
      soil_humidity: another_soil_humidity_sensor_value,
      enabled: another_soil_humidity_sensor_enabled
    )
  }
end

context 'when the other soil humidity sensor is enabled' do
  let(:another_soil_humidity_sensor_enabled) { true }

  context 'when the soil humidity sensor is below the threshold' do
    let(:soil_humidity_sensor_value) { humidity_threshold - 1 }

    context 'when the other soil humidity sensor pass the threshold' do
      let(:another_soil_humidity_sensor_value) { humidity_threshold + 1 }

      it 'does not change the lighting system state' do
        was_active = irrigation_system.active?

        expect(irrigation_system.reload.active?).to eq was_active
      end
    end

    context 'when the other humidity sensor is below the threshold too' do
      let(:another_soil_humidity_sensor_value) { humidity_threshold - 1 }

      it 'activates the lighting system' do
        expect(irrigation_system.reload).to be_active
      end
    end
  end

  context 'when the soil humidity sensor pass the threshold' do
    let(:soil_humidity_sensor_value) { humidity_threshold + 1 }

    context 'when the other humidity sensor is below the threshold too' do
      let(:another_soil_humidity_sensor_value) { humidity_threshold - 1 }

      it 'does not change the lighting system state' do
        was_active = irrigation_system.active?

        expect(irrigation_system.reload.active?).to eq was_active
      end
    end

    context 'when the other soil humidity sensor pass the threshold' do
      let(:another_soil_humidity_sensor_value) { humidity_threshold + 1 }

      it 'inactivates the lighting system' do
        expect(irrigation_system.reload).to be_inactive
      end
    end
  end
end

context 'when the other soil humidity sensor is disabled' do
  let(:another_soil_humidity_sensor_enabled) { false }

  context 'when the soil humidity sensor pass the threshold' do
    let(:soil_humidity_sensor_value) { humidity_threshold + 1 }

    context 'when the other soil humidity sensor is below the threshold' do
```

```
    let(:another_soil_humidity_sensor_value) { humidity_threshold - 1 }

    it 'inactivates the lighting system' do
      expect(irrigation_system.reload).to be_inactive
    end
  end

  context 'when the other soil humidity sensor pass the threshold too' do
    let(:another_soil_humidity_sensor_value) { humidity_threshold + 1 }

    it 'inactivates the lighting system' do
      expect(irrigation_system.reload).to be_inactive
    end
  end

  context 'when the soil humidity sensor is below the threshold' do
    let(:soil_humidity_sensor_value) { humidity_threshold - 1 }

    context 'when the other humidity sensor is below the threshold too' do
      let(:another_soil_humidity_sensor_value) { humidity_threshold - 1 }

      it 'activates the lighting system' do
        expect(irrigation_system.reload).to be_active
      end
    end

    context 'when the other soil humidity sensor pass the threshold' do
      let(:another_soil_humidity_sensor_value) { humidity_threshold + 1 }

      it 'activates the lighting system' do
        expect(irrigation_system.reload).to be_active
      end
    end
  end
end
```

Los tests mostrados como ejemplo son el resultado de la aplicación de la técnica de desarrollo guiado por tests para modelar el comportamiento de reacción del sistema frente a la lectura de un cambio en los datos recolectados por los sensores de luminosidad. El modelo logrado en consecuencia se detalla a continuación:

```
class LuminositySensorEventHandler < DeviceEventHandler

  def receive(luminosity_sensor, luminosity_sensor_value)
    luminosity_sensor.luminosity! luminosity_sensor_value
    activate_or_inactivate_associated_lighting_systems(luminosity_sensor)
    rescue ActiveRecord::RecordInvalid => e
      puts "Invalid luminosity value #{e.message}"
    end

  private

  def activate_or_inactivate_associated_lighting_systems(luminosity_sensor)
    enabled_lighting_systems(luminosity_sensor).each do |lighting_system|
      on_activate_block = proc { Notification.lighting_system_activated }
      on_inactivate_block = proc { Notification.lighting_system_deactivated }
      lighting_system.activate_or_inactivateDependingOnSensors(
        on_activate_block,
        on_inactivate_block
      )
    end
  end

  def enabled_lighting_systems(luminosity_sensor)
    luminosity_sensor.lighting_systems.enabled
  end
end
```

Como se puede ver, cuando se recibe un nuevo valor de luminosidad para un nuevo sensor de luminosidad, se actualiza el valor del objeto y también se notifican los sistemas de iluminación asociados.

Usuarios

El servidor central, además de permitirnos comunicarnos con la arquitectura, es el encargado de llevar a cabo la autenticación de usuarios. El servidor consta con una lista de usuarios registrados a los cuales se les permite gestionar el sistema. Es por eso que mediante credenciales compuestas por una dirección de correo electrónico y una contraseña de 16 caracteres, el servidor debe autenticar a esos usuarios de forma tal que se mantenga la seguridad de que sólo aquellos usuarios autorizados son los que gestionan la aplicación.

Para llevar a cabo el modelo de los usuarios en la aplicación se utilizó la gema de Ruby llamada Devise⁸. Esta gema es una solución flexible para autenticación en aplicaciones Rails.



Figura 21. Logo de Devise

⁸ Visitar el sitio web de la gema Devise en GitHub para más información - github.com/plataformatec/devise

Es una gema muy utilizada y muy recomendada por la comunidad. Es muy potente e incluye una gran cantidad de funcionalidades. Te permite tener implementar la autenticación de múltiples modelos a la vez, no hace falta tener una única clase de usuarios. Además está compuesta por diez módulos que nos proveen toda la funcionalidad necesaria para el manejo de usuarios:

Registrable

Permite gestionar el proceso de registración de usuarios, permitiendo también que se pueda editar o borrar la información de las cuentas de usuario.

Autenticación

Se encarga de cifrar y almacenar las passwords en la base de datos y de la validación de autenticidad al momento de registrarse o loggearse en la aplicación.

Confirmación

Permite gestionar el manejo de mails para incluir las instrucciones de confirmación de cuentas de usuarios cuando se registran, además permite verificar si las cuentas asociadas se encuentran confirmadas.

Restauración de passwords

Permite restaurar passwords por medio de pasos enviados por email, asegurando la protección de la información sensible.

Recordar

Se encarga de manejar la generación y limpieza de tokens para recordar a los usuarios registrados o loggeados por medio del uso de cookies.

Tracking

También incluye un tracking del cantidad de veces que se loggeó un usuario, indicando el instante en que se realizó y la dirección IP proveniente.

Sesiones

Se encarga de trabajar con sesiones que expiran luego de que no hayan estado activas por una cantidad específica de tiempo.

Validaciones

Provee validaciones especiales para campos de tipo email y password, son opcionales y pueden ser customisables.

Bloqueo

Permite gestionar el bloqueo automático de cuentas luego de sobrepasar una cantidad específica de intentos fallidos de acceso a una cuenta. Los usuarios pueden desbloquearse siguiendo unos pasos enviados por mail luego de un cierto período de tiempo.

OmniAuth

Agrega soporte para OmniAuth, que es el protocolo actualmente utilizado por la mayoría de las redes sociales para autenticarse a través de sus usuarios. Es implementado por Facebook, Twitter, Instagram, Gmail, y muchos más.

Para esta implementación se utilizó Devise tal como se recomienda con la salvedad de que los usuarios sólo pueden ser creados manualmente una vez que estamos ingresados en el sistema, no es posible registrarse como es común en las aplicaciones web. Esto es así para garantizar que sólo aquellos usuarios creados por los responsables de gestionar el sistema de IntelliHome, son quienes podrán gestionar los ambientes y artefactos del hogar.

Protocolos

Introducción

El servidor central implementa dos protocolos, un protocolo JSON a través del cual expone una API pública para interactuar idealmente con las interfaces web o mobile, y otro protocolo MQTT a través de la cual interactúa de manera privada con los dispositivos cliente Raspberry Pi. A continuación describiremos cada uno de ellos

JSON

JSON o JavaScript Object Notation es un formato estándar de texto legible para la transmisión de objetos según pares de clave-valor. Es el formato de datos más común utilizado para las comunicaciones entre los browsers y los servidores de contenido web. JSON reemplazó notablemente a XML, principalmente porque es mucho más sencillo de generar, de entender y de escribir analizadores sintácticos que puedan parsearlos.

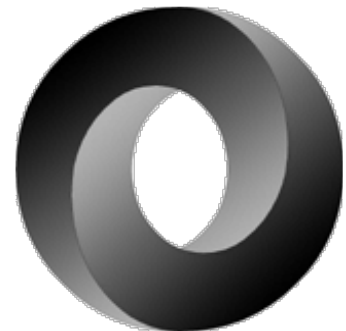


Figura 22. Logo de JSON

El servidor central expone una API JSON para poder comunicarnos e interactuar con él. Dicha interfaz nos abstrae de la compleja arquitectura permitiéndonos configurarla y controlarla. Esta API se describe a continuación:

Usuarios

A continuación se detallan las rutas de la API que nos permiten llevar a cabo el manejo de usuarios.

GET /users

Esta ruta es utilizada para solicitar información sobre todos los usuarios. Esta acción nos devolverá una vista JSON con los datos de todos los usuarios:

```
{
  "users": [
    {
      "id": 1,
      "first_name": "Facundo",
      "last_name": "Spagnuolo",
      "email": "facundo_spagnuolo@intellihome.com",
      "member_since": "02/04/2016"
    },
    {
      "id": 2,
      "first_name": "Delfina",
      "last_name": "Polito",
      "email": "delfina_polito@intellihome.com",
      "member_since": "05/06/2016"
    }
  ]
}
```

GET /users/:id

Esta ruta es utilizada para solicitar información sobre algún usuario específico. Para referirnos al usuarios debemos indicar el id del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del usuario solicitado:

```
{
  "id": 1,
  "first_name": "Facundo",
  "last_name": "Spagnuolo",
  "email": "facundo_spagnuolo@intellihome.com",
  "member_since": "02/04/2016"
}
```

POST /users

Ruta utilizada para crear una nueva cuenta de usuario. Debemos indicar el nombre (first_name), el apellido (last_name), su dirección de correo (email), su clave

(password) y su confirmación de clave (password_confirmation) de la siguiente manera:

```
{
  "user": {
    "first_name": "Facundo",
    "last_name": "Spagnuolo",
    "email": "facundo_spagnuolo@intellihome.com",
    "password": "1234567812345678",
    "password_confirmation": "1234567812345678",
  }
}
```

Esta acción nos devolverá una vista JSON con los datos del usuario creado:

```
{
  "id": 1,
  "first_name": "Facundo",
  "last_name": "Spagnuolo",
  "email": "facundo_spagnuolo@intellihome.com",
  "member_since": "02/04/2016"
}
```

PATCH /users/:id

Esta ruta es utilizada para actualizar cierta información sobre algún usuario específico. Para referirnos al usuario debemos indicar el `id` del mismo en la propia ruta. Los atributos que podemos modificar son el nombre (`first_name`), el apellido (`last_name`) o el correo electrónico (`email`) del usuario, de la siguiente manera:

```
{
  "user": {
    "first_name": "Carlos",
    "email": "carlos_spagnuolo@intellihome.com",
  }
}
```

Esta acción nos devolverá una vista JSON con los datos del usuario actualizado:


```
{
  "id": 1,
  "first_name": "Carlos",
  "last_name": "Spagnuolo",
  "email": "carlos_spagnuolo@intellihome.com",
  "member_since": "02/04/2016"
}
```

POST /login

Ruta utilizada para loggear una cuenta de usuario. Debemos indicar el correo electrónico (email) y su clave (password) de la siguiente manera:

```
{
  "user": {
    "email": "facundo_spagnuolo@intellihome.com",
    "password": "1234567812345678",
  }
}
```

DELETE /logout

Esta ruta es utilizada para desloguear la sesión actual del usuario.

SENSORES

A continuación se detallan las rutas de la API que nos permiten llevar a cabo el manejo de todos los tipos de sensores:

GET /motion_sensors

Esta ruta es utilizada para solicitar información sobre todos los sensores de movimiento. Esta acción nos devolverá una vista JSON con los datos de todos los sensores de movimiento:

```
{
  "motion_sensors": [
    {
      "id": 1,
      "name": "Sensor sur",
      "description": "Alarma perimetral",
      "room_name": "Jardín",
      "enabled": true,
      "created_at": "02/04/2016"
    },
    {
      "id": 2,
      "name": "Sensor norte",
      "description": "Alarma perimetral",
      "room_name": "Jardín",
      "enabled": true,
      "created_at": "02/04/2016"
    }
  ]
}
```

GET /motion_sensors/:id

Esta ruta es utilizada para solicitar información sobre algún sensor de movimiento específico. Para referirnos al sensor debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del sensor solicitado:

```
{
  "id": 1,
  "name": "Sensor sur",
  "description": "Alarma perimetral",
  "room_name": "Jardín",
  "enabled": true,
  "created_at": "02/04/2016"
}
```

PUT /motion_sensors/:id

Esta ruta es utilizada para actualizar cierta información sobre algún sensor de movimiento específico. Para referirnos al sensor debemos indicar el `id` del mismo en la propia ruta. Los atributos que podemos modificar son el nombre (`name`), la

descripción (`description`) o la habitación a la cual pertenece (`room_id`) del sensor, de la siguiente manera:

```
{
  "motion_sensor": {
    "name": "Sensor de la entrada",
    "room_id": 4,
  }
}
```

Esta acción nos devolverá una vista JSON con los datos del sensor actualizado:

```
{
  "id": 1,
  "name": "Sensor de la entrada",
  "description": "Alarma perimetral",
  "room_name": "Entrada",
  "enabled": true,
  "created_at": "02/04/2016"
}
```

POST /motion_sensors/:id/enable

Esta ruta es utilizada para habilitar un sensor de movimiento específico. Para referirnos al sensor debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del sensor actualizado.

POST /motion_sensors/:id/disable

Esta ruta es utilizada para inhabilitar un sensor de movimiento específico. Para referirnos al sensor debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del sensor actualizado.

GET /soil_humidity_sensors

Esta ruta es utilizada para solicitar información sobre todos los sensores de humedad del suelo. Esta acción nos devolverá una vista JSON con los datos de todos los sensores de humedad del suelo:

```
{
  "soil_humidity_sensors": [
    {
      "id": 1,
      "name": "Sensor de los naranjos",
      "description": "Riego automático",
      "room_name": "Jardín",
      "enabled": true,
      "created_at": "02/04/2016"
    },
    {
      "id": 2,
      "name": "Sensor de los manzanos",
      "description": "Riego automático",
      "room_name": "Jardín",
      "enabled": true,
      "created_at": "02/04/2016"
    }
  ]
}
```

GET /soil_humidity_sensors/:id

Esta ruta es utilizada para solicitar información sobre algún sensor de humedad del suelo específico. Para referirnos al sensor debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del sensor solicitado:

```
{
  "id": 1,
  "name": "Sensor de los naranjos",
  "description": "Riego automático",
  "room_name": "Jardín",
  "enabled": true,
  "created_at": "02/04/2016"
}
```

PUT /soil_humidity_sensors/:id

Esta ruta es utilizada para actualizar cierta información sobre algún sensor de humedad del suelo específico. Para referirnos al sensor debemos indicar el `id` del

mismo en la propia ruta. Los atributos que podemos modificar son el nombre (name), la descripción (description) o la habitación a la cual pertenece (room_id) del sensor, de la siguiente manera:

```
{
  "soil_humidity_sensor": {
    "name": "Sensor de los manzanos",
    "room_id": 4,
  }
}
```

Esta acción nos devolverá una vista JSON con los datos del sensor actualizado:

```
{
  "id": 1,
  "name": "Sensor de los manzanos",
  "description": "Riego automático",
  "room_name": "Entrada",
  "enabled": true,
  "created_at": "02/04/2016"
}
```

POST /soil_humidity_sensors/:id/enable

Esta ruta es utilizada para habilitar un sensor de humedad del suelo específico. Para referirnos al sensor debemos indicar el id del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del sensor actualizado.

POST /soil_humidity_sensors/:id/disable

Esta ruta es utilizada para inhabilitar un sensor de humedad del suelo específico. Para referirnos al sensor debemos indicar el id del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del sensor actualizado.

GET /luminosity_sensors

Esta ruta es utilizada para solicitar información sobre todos los sensores de luminosidad. Esta acción nos devolverá una vista JSON con los datos de todos los sensores de luminosidad:

```
{
  "luminosity_sensors": [
    {
      "id": 1,
      "name": "Sensor este",
      "description": "Sistema de luz automático",
      "room_name": "Jardín",
      "enabled": true,
      "created_at": "02/04/2016"
    },
    {
      "id": 2,
      "name": "Sensor oeste",
      "description": "Sistema de luz automático",
      "room_name": "Jardín",
      "enabled": true,
      "created_at": "02/04/2016"
    }
  ]
}
```

GET /luminosity_sensors/:id

Esta ruta es utilizada para solicitar información sobre algún sensor luminosidad específico. Para referirnos al sensor debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del sensor solicitado:

```
{
  "id": 1,
  "name": "Sensor este",
  "description": "Sistema de luz automático",
  "room_name": "Jardín",
  "enabled": true,
  "created_at": "02/04/2016"
}
```

PUT /luminosity_sensors/:id

Esta ruta es utilizada para actualizar cierta información sobre algún sensor de luminosidad específico. Para referirnos al sensor debemos indicar el `id` del mismo en la propia ruta. Los atributos que podemos modificar son el nombre (`name`), la

descripción (*description*) o la habitación a la cual pertenece (*room_id*) del sensor, de la siguiente manera:

```
{
  "luminosity_sensor": {
    "name": "Sensor de la entrada",
    "room_id": 4,
  }
}
```

Esta acción nos devolverá una vista JSON con los datos del sensor actualizado:

```
{
  "id": 1,
  "name": "Sensor de la entrada",
  "description": "Sistema de luz automático",
  "room_name": "Entrada",
  "enabled": true,
  "created_at": "02/04/2016"
}
```

POST /luminosity_sensors/:id/enable

Esta ruta es utilizada para habilitar un sensor de luminosidad específico. Para referirnos al sensor debemos indicar el *id* del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del sensor actualizado.

POST /luminosity_sensors/:id/disable

Esta ruta es utilizada para inhabilitar un sensor de luminosidad específico. Para referirnos al sensor debemos indicar el *id* del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del sensor actualizado.

ARTEFACTOS

A continuación se detallan las rutas de la API que nos permiten llevar a cabo el manejo de todos los tipos de artefactos de un hogar:

GET /lights

Esta ruta es utilizada para solicitar información sobre todas las luces del hogar. Esta acción nos devolverá una vista JSON con los datos de todas las luces:

```
{
  "lights": [
    {
      "id": 1,
      "name": "Velador derecho",
      "description": "40 W",
      "room_name": "Habitación",
      "switched_on": true,
      "created_at": "02/04/2016"
    },
    {
      "id": 2,
      "name": "Velador izquierdo",
      "description": "60 W",
      "room_name": "Habitación",
      "switched_on": false,
      "created_at": "02/04/2016"
    }
  ]
}
```

GET /lights/:id

Esta ruta es utilizada para solicitar información sobre alguna luz específica. Para referirnos a la luz debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos de la luz solicitada:

```
{
  "id": 1,
  "name": "Velador derecho",
  "description": "40 W",
  "room_name": "Habitación",
  "switched_on": true,
  "created_at": "02/04/2016"
}
```


PUT /lights/:id

Esta ruta es utilizada para actualizar cierta información sobre alguna luz específica. Para referirnos a la luz debemos indicar el `id` del mismo en la propia ruta. Los atributos que podemos modificar son el nombre (`name`), la descripción (`description`) o la habitación a la cual pertenece (`room_id`) de la luz, de la siguiente manera:

```
{
  "light": {
    "name": "Velador del escritorio",
    "room_id": 3,
  }
}
```

Esta acción nos devolverá una vista JSON con los datos de la luz actualizada:

```
{
  "id": 1,
  "name": "Velador del escritorio",
  "description": "40 W",
  "room_name": "Living",
  "switched_on": true,
  "created_at": "02/04/2016"
}
```

POST /lights/:id/switch_on

Esta ruta es utilizada para encender una luz específica. Para referirnos a la luz debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos de la luz actualizada.

POST /lights/:id/switch_off

Esta ruta es utilizada para apagar una luz específica. Para referirnos a la luz debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos de la luz actualizada.

GET /tvs

Esta ruta es utilizada para solicitar información sobre todas las televisiones del hogar. Esta acción nos devolverá una vista JSON con los datos de todas las televisiones:

```
{
  "tvs": [
    {
      "id": 1,
      "name": "TV Habitación",
      "description": "Samsung 42'",
      "room_name": "Habitación",
      "turned_on": false,
      "created_at": "02/04/2016"
    },
    {
      "id": 2,
      "name": "TV Living",
      "description": "Samsung 50'",
      "room_name": "Living",
      "turned_on": false,
      "created_at": "02/04/2016"
    }
  ]
}
```

GET /tvs/:id

Esta ruta es utilizada para solicitar información sobre alguna televisión específica. Para referirnos a la televisión debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos de la televisión solicitada:

```
{
  "id": 1,
  "name": "TV Habitación",
  "description": "Samsung 42'",
  "room_name": "Habitación",
  "turned_on": true,
  "created_at": "02/04/2016"
}
```

PUT /tvs/:id

Esta ruta es utilizada para actualizar cierta información sobre alguna televisión específica. Para referirnos a la televisión debemos indicar el `id` del mismo en la propia ruta. Los atributos que podemos modificar son el nombre (`name`), la descripción (`description`) o la habitación a la cual pertenece (`room_id`) de la televisión, de la siguiente manera:

```
{
  "tv": {
    "name": "TV del escritorio",
    "room_id": 3,
  }
}
```

Esta acción nos devolverá una vista JSON con los datos de la televisión actualizada:

```
{
  "id": 1,
  "name": "TV del escritorio",
  "description": "Samsung 42'",
  "room_name": "Living",
  "turned_on": true,
  "created_at": "02/04/2016"
}
```

POST /tvs/:id/turn_on

Esta ruta es utilizada para encender una televisión específica. Para referirnos a la televisión debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos de la televisión actualizada.

POST /tvs/:id/turn_off

Esta ruta es utilizada para apagar una televisión específica. Para referirnos a la televisión debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos de la televisión actualizada.

GET /valves

Esta ruta es utilizada para solicitar información sobre todas las canillas del hogar.

Esta acción nos devolverá una vista JSON con los datos de todas las canillas:

```
{
  "valves": [
    {
      "id": 1,
      "name": "Canilla del Baño",
      "description": "Agua fría",
      "room_name": "Habitación",
      "open": false,
      "created_at": "02/04/2016"
    },
    {
      "id": 2,
      "name": "Canilla del jardín",
      "description": "Agua fría",
      "room_name": "Jardín",
      "open": true,
      "created_at": "02/04/2016"
    }
  ]
}
```

GET /valves/:id

Esta ruta es utilizada para solicitar información sobre alguna canilla específica. Para referirnos a la canilla debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos de la canilla solicitada:

```
{
  "id": 1,
  "name": "Canilla del Baño",
  "description": "Agua fría",
  "room_name": "Habitación",
  "open": false,
  "created_at": "02/04/2016"
}
```

PUT /valves/:id

Esta ruta es utilizada para actualizar cierta información sobre alguna canilla específica. Para referirnos a la canilla debemos indicar el `id` del mismo en la propia ruta. Los atributos que podemos modificar son el nombre (`name`), la descripción (`description`) o la habitación a la cual pertenece (`room_id`) de la canilla, de la siguiente manera:

```
{
  "valve": {
    "name": "Canilla de la cocina",
    "room_id": 5,
  }
}
```

Esta acción nos devolverá una vista JSON con los datos de la canilla actualizada:

```
{
  "id": 1,
  "name": "Canilla de la cocina",
  "description": "Agua fría",
  "room_name": "Cocina",
  "open": false,
  "created_at": "02/04/2016"
}
```

POST /valves/:id/open

Esta ruta es utilizada para abrir una canilla específica. Para referirnos a la canilla debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos de la canilla actualizada.

POST /valves/:id/close

Esta ruta es utilizada para cerrar una canilla específica. Para referirnos a la canilla debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos de la canilla actualizada.

SISTEMAS

A continuación se detallan las rutas de la API que nos permiten llevar a cabo el manejo de todos los tipos de sistemas:

GET /alarms

Esta ruta es utilizada para solicitar información sobre todas las alarmas. Esta acción nos devolverá una vista JSON con los datos de todas las alarmas:

```
{
  "alarms": [
    {
      "id": 1,
      "name": "Alarma perimetral",
      "description": "Exterior",
      "enabled": true,
      "active": false,
      "motion_sensors": [...],
      "created_at": "02/04/2016"
    },
    {
      "id": 2,
      "name": "Alarma interna",
      "description": "Interior",
      "enabled": true,
      "active": false,
      "motion_sensors": [...],
      "created_at": "02/04/2016"
    }
  ]
}
```

GET /alarms/:id

Esta ruta es utilizada para solicitar información sobre alguna alarma específica. Para referirnos a la alarma debemos indicar el id del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos de la alarma solicitada:

```
{
  "id": 1,
  "name": "Alarma interna",
  "description": "Interior",
  "enabled": true,
  "active": false,
  "motion_sensors": [...],
  "created_at": "02/04/2016"
}
```

POST /alarms

Ruta utilizada para crear una nueva alarma. Debemos indicar el nombre (name), la descripción (description), y los sensores de movimiento que la controlan (motion_sensors_ids) de la siguiente manera:

```
{
  "alarm": {
    "name": "Alarma Interna",
    "description": "Interior",
    "motion_sensors_ids": [1,4,5]
  }
}
```

Esta acción nos devolverá una vista JSON con los datos de la alarma creada:

```
{
  "id": 1,
  "name": "Alarma interna",
  "description": "Interior",
  "enabled": true,
  "active": false,
  "motion_sensors": [...],
  "created_at": "02/04/2016"
}
```

PUT /alarms/:id

Esta ruta es utilizada para actualizar cierta información sobre alguna canilla específica. Para referirnos a la canilla debemos indicar el id del mismo en la propia

ruta. Los atributos que podemos modificar son el nombre (`name`), la descripción (`description`), y los sensores de movimiento que la controlan (`motion_sensors_ids`) de la siguiente manera:

```
{
  "alarm": {
    "name": "Alarma del interior del hogar",
    "description": "Interior",
    "motion_sensors_ids": [1]
  }
}
```

Esta acción nos devolverá una vista JSON con los datos de la alarma actualizada:

```
{
  "id": 1,
  "name": "Alarma del interior del hogar",
  "description": "Interior",
  "enabled": true,
  "active": false,
  "motion_sensors": [...],
  "created_at": "02/04/2016"
}
```

POST /alarms/:id/enable

Esta ruta es utilizada para habilitar una alarma específica. Para referirnos a la alarma debemos indicar el `id` de la misma en la propia ruta. Esta acción nos devolverá una vista JSON con los datos de la alarma actualizada.

POST /alarms/:id/disable

Esta ruta es utilizada para inhabilitar una alarma específica. Para referirnos a la alarma debemos indicar el `id` de la misma en la propia ruta. Esta acción nos devolverá una vista JSON con los datos de la alarma actualizada.

POST /alarms/:id/activate

Esta ruta es utilizada para activar una alarma, es decir, hacerla sonar. Para referirnos a la alarma debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos de la alarma actualizada.

POST /alarms/:id/deactivate

Esta ruta es utilizada para desactivar una alarma específica, es decir, hacer que pare de sonar. Para referirnos a la alarma debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos de la alarma actualizada.

GET /irrigation_systems

Esta ruta es utilizada para solicitar información sobre todos los sistemas de riego. Esta acción nos devolverá una vista JSON con los datos de todos los sistemas de riego:

```
{
  "irrigation_systems": [
    {
      "id": 1,
      "name": "Sistema de riego de verduras",
      "description": "Jardín",
      "enabled": true,
      "active": false,
      "valves": [...],
      "soil_humidity_sensors": [...],
      "created_at": "02/04/2016"
    },
    {
      "id": 2,
      "name": "Sistema de riego de frutas",
      "description": "Jardín",
      "enabled": true,
      "active": true,
      "valves": [...],
      "soil_humidity_sensors": [...],
      "created_at": "02/04/2016"
    }
  ]
}
```

GET /irrigation_systems/:id

Esta ruta es utilizada para solicitar información sobre algún sistema de riego específico. Para referirnos al sistema de riego debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del mismo sistema de riego solicitado:

```
{
  "id": 1,
  "name": "Sistema de riego de verduras",
  "description": "Jardín",
  "enabled": true,
  "active": false,
  "valves": [...],
  "soil_humidity_sensors": [...],
  "created_at": "02/04/2016"
}
```

POST /irrigation_systems

Ruta utilizada para crear un nuevo sistema de riego. Debemos indicar el nombre (`name`), la descripción (`description`), los sensores de humedad del suelo que lo controlan (`soil_humidity_sensors_ids`) y las válvulas que controlan (`valves_ids`) de la siguiente manera:

```
{
  "irrigation_system": {
    "name": "Sistema de riego de los tomates",
    "description": "Jardín",
    "valves_ids": [4,5]
    "soil_humidity_sensors_ids": [1,2]
  }
}
```

Esta acción nos devolverá una vista JSON con los datos del sistema de riego creado:

```
{
  "id": 1,
  "name": "Sistema de riego de los tomates",
  "description": "Jardín",
  "enabled": true,
  "active": false,
  "valves": [...]
  "soil_humidity_sensors": [...]
  "created_at": "02/04/2016"
}
```

PUT /irrigation_systems/:id

Esta ruta es utilizada para actualizar cierta información sobre alguna canilla específica. Para referirnos a la canilla debemos indicar el `id` del mismo en la propia ruta. Los atributos que podemos modificar son el nombre (`name`), la descripción (`description`), y los sensores de humedad del suelo que lo controlan (`soil_humidity_sensors_ids`) y las válvulas que controlan (`valves_ids`) de la siguiente manera:

```
{
  "irrigation_system": {
    "name": "Alarma del interior del hogar",
    "description": "Interior",
    "valves_ids": [4,5],
    "soil_humidity_sensors_ids": [1]
  }
}
```

Esta acción nos devolverá una vista JSON con los datos del sistema de riego actualizado:

```
{
  "id": 1,
  "name": "Sistema de riego de los tomates",
  "description": "Jardín",
  "enabled": true,
  "active": false,
  "valves": [...]
  "soil_humidity_sensors": [...]
  "created_at": "02/04/2016"
}
```

POST /irrigation_systems/:id/enable

Esta ruta es utilizada para habilitar un sistema de riego específico. Para referirnos al sistema de riego, debemos indicar el `id` de la misma en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del sistema de riego actualizado.

POST /irrigation_systems/:id/disable

Esta ruta es utilizada para inhabilitar un sistema de riego específico. Para referirnos al sistema de riego, debemos indicar el `id` de la misma en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del sistema de riego actualizado.

POST /irrigation_systems/:id/activate

Esta ruta es utilizada para activar un sistema de riego específico, es decir, abrir todas las canillas asociadas. Para referirnos al sistema de riego, debemos indicar el `id` de la misma en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del sistema de riego actualizado.

POST /irrigation_systems/:id/deactivate

Esta ruta es utilizada para desactivar un sistema de riego específico, es decir, cerrar todas las canillas asociadas. Para referirnos al sistema de riego, debemos indicar el `id` de la misma en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del sistema de riego actualizado.

GET /lighting_systems

Esta ruta es utilizada para solicitar información sobre todos los sistemas de iluminación. Esta acción nos devolverá una vista JSON con los datos de todos los sistemas de iluminación:

```
{
  "lighting_systems": [
    {
      "id": 1,
      "name": "Sistema de iluminación perimetral",
      "description": "Externo",
      "enabled": true,
      "active": false,
      "valves": [...],
      "soil_humidity_sensors": [...],
      "created_at": "02/04/2016"
    },
    {
      "id": 2,
      "name": "Sistema de iluminación interno",
      "description": "Interno",
      "enabled": true,
      "active": true,
      "lights": [...],
      "soil_humidity_sensors": [...],
      "created_at": "02/04/2016"
    },
  ],
}
```

GET /lighting_systems/:id

Esta ruta es utilizada para solicitar información sobre algún sistema de iluminación específico. Para referirnos al sistema de iluminación debemos indicar el `id` del mismo en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del mismo sistema de iluminación solicitado:

```
{
  "id": 1,
  "name": "Sistema de iluminación perimetral",
  "description": "Exterior",
  "enabled": true,
  "active": false,
  "lights": [...],
  "luminosity_sensors": [...],
  "created_at": "02/04/2016"
}
```

POST /lighting_systems

Ruta utilizada para crear un nuevo sistema de iluminación. Debemos indicar el nombre (name), la descripción (description), los sensores de humedad del suelo que lo controlan (luminosity_sensors_ids) y las válvulas que controlan (lights_ids) de la siguiente manera:

```
{
  "irrigation_system": {
    "name": "Sistema de iluminación perimetral",
    "description": "Jardín",
    "lights_ids": [4,5]
    "luminosity_sensors_ids": [1,2]
  }
}
```

Esta acción nos devolverá una vista JSON con los datos del sistema de iluminación creado:

```
{
  "id": 1,
  "name": "Sistema de iluminación perimetral",
  "description": "Jardín",
  "enabled": true,
  "active": false,
  "lights": [...]
  "luminosity_sensors": [...]
  "created_at": "02/04/2016"
}
```

PUT /lighting_systems/:id

Esta ruta es utilizada para actualizar cierta información sobre alguna canilla específica. Para referirnos a la canilla debemos indicar el id del mismo en la propia ruta. Los atributos que podemos modificar son el nombre (name), la descripción (description), y los sensores de humedad del suelo que lo controlan (luminosity_sensors_ids) y las válvulas que controlan (lights_ids) de la siguiente manera:

```
{
  "lighting_system": {
    "name": "Sistema de iluminación del jardín",
    "description": "Jardín",
    "lights_ids": [1],
    "luminosity_sensors_ids": [2,5]
  }
}
```

Esta acción nos devolverá una vista JSON con los datos del sistema de iluminación actualizado:

```
{
  "id": 1,
  "name": "Sistema de iluminación del jardín",
  "description": "Jardín",
  "enabled": true,
  "active": false,
  "lights": [...]
  "luminosity_sensors": [...]
  "created_at": "02/04/2016"
}
```

POST /lighting_systems/:id/enable

Esta ruta es utilizada para habilitar un sistema de iluminación específico. Para referirnos al sistema de iluminación, debemos indicar el `id` de la misma en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del sistema de iluminación actualizado.

POST /lighting_systems/:id/disable

Esta ruta es utilizada para inhabilitar un sistema de iluminación específico. Para referirnos al sistema de iluminación, debemos indicar el `id` de la misma en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del sistema de iluminación actualizado.

POST /lighting_systems/:id/activate

Esta ruta es utilizada para activar un sistema de iluminación específico, es decir, abrir todas las canillas asociadas. Para referirnos al sistema de iluminación, debemos indicar el `id` de la misma en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del sistema de iluminación actualizado.

POST /lighting_systems/:id/deactivate

Esta ruta es utilizada para desactivar un sistema de iluminación específico, es decir, apagar todas las luces asociadas. Para referirnos al sistema de iluminación, debemos indicar el `id` de la misma en la propia ruta. Esta acción nos devolverá una vista JSON con los datos del sistema de iluminación actualizado.

ERRORES

En todos los casos mencionados, si llegase a ocurrir un error del lado del servidor, el mismo responderá con un mensaje único indicando el estado HTTP del mismo con el siguiente formato:

```
{
  "errors": [
    "Mensaje de error 1",
    "Mensaje de error 2",
    ...
    "Mensaje de error N"
  ]
}
```

AUTENTICACIÓN

Para llevar a cabo la autenticación en todo el protocolo, la implementación de la API JSON utiliza un mecanismo de tokens. Para ello, una vez que un usuario se logea, el servidor envía un token en la cabecera que será requerido en próximo request. Para incluirlo se deberá asignarlo al campo `Authorization` en la cabecera del próximo request. Este token cambia en cada respuesta, es decir, es único por mensaje.

MQTT

MQTT o formalmente *Message Queuing Telemetry Transport* es un estándar ISO que define un protocolo de mensajería liviano publicador-subscriptor, que funciona montado



Figura 23. Logo de la organización MQTT

sobre el protocolo TCP/IP. Fue diseñado para conexiones remotas donde se requiere poca cantidad de código para implementarlo y donde el ancho de banda es limitado. La primera versión de este protocolo fue desarrollado en IBM en el año 1999.

Los protocolos de mensajería estilo publicador-subscriptor requieren de un broker. Este es el responsable de distribuir los mensajes a los clientes interesados. Para distinguirlos suelen implementarse tópicos o canales, a los que cada cliente puede subscribirse para indicar que está interesado en recibir los mensajes que allí se posteen.

Existen un montón de implementaciones de servidores MQTT, para esta implementación utilizamos Mosquitto en los clientes Raspberry, y una implementación casera en Ruby realizada por Nicholas Humfrey, un desarrollador Ruby de Inglaterra.

SENSORES

Una parte del protocolo fue implementada para enviar información a cerca de los sensores, es decir, para llevar a cabo la comunicación desde los clientes hacia el servidor central. Para ello, se decidió utilizar un tópico por cada tipo de sensor sobre el cual se reporta información. Por lo tanto, servidor posee un servicio por cada tipo de sensor escuchando esos tópicos, de forma tal que se enterará cada vez que algún sensor envíe una información nueva. De esta manera el servidor puede saber que hacer con la información que se recolecta de cada uno de los sensores, como

por ejemplo notificarlo a cualquiera de los sistemas correspondiente. La convención que llevamos a cabo para implementar los mensajes del protocolo es algo similar a una API REST: **rooms/:room_id/:type_of_sensor/:sensor_id/:value**

Donde **room_id** es el identificador de ambiente que comparten ambos, **type_of_sensor** es el tipo de sensor sobre el que se está informando el estado pudiendo ser **motion_sensors**, **luminosity_sensors**, **humidity_sensors**, **soil_humidity_sensors** o **temperature_sensors**. Luego **sensor_id** es el identificador del sensor que comparten el servidor y el cliente y **value** es el estado que se está reportando sobre la lectura del sensor.

rooms/:room_id/motion_sensors/:motion_sensor_id/ON

Esta ruta sirve para indicar que un sensor de movimiento ha detectado la presencia de movimiento.

rooms/:room_id/motion_sensors/:motion_sensor_id/OFF

Esta ruta sirve para indicar que un sensor de movimiento ha dejado de detectar la presencia de movimiento.

rooms/:room_id/luminosity_sensors/:luminosity_sensor_id/:luminosity

Esta ruta sirve para indicar el valor de la intensidad de luz que un sensor de luminosidad está detectando. Este valor es un dato numérico entre 0 y 1000, donde 0 implica oscuridad y 1000 implica la mayor cantidad de luz que el sensor puede captar.

rooms/:room_id/humidity_sensors/:humidity_sensor_id/:humidity

Esta ruta sirve para indicar el valor de humedad que un sensor de humedad está detectando. Este valor es un dato porcentual entre 0% y 100%.

rooms/:room_id/temperature_sensors/:temperature_sensor_id/:temperature

Esta ruta sirve para indicar el valor de temperatura que un sensor de temperatura está detectando. Este valor es un dato numérico en grados Celsius.

rooms/:room_id/soil_humidity_sensors/:soil_humidity_sensor_id/:humidity

Esta ruta sirve para indicar el valor de humedad del suelo que un sensor de humedad del suelo está detectando. Este valor es un dato numérico entre 0 y 1000, donde 0 implica seco (aire) y 1000 implica totalmente húmedo (agua).

ACTUADORES

Por otro lado, para que el servidor se comuniquen con los clientes Raspberry y pueda controlar ciertos artefactos a través de los actuadores, se decidió utilizar un tópico por cada ambiente del hogar, de forma tal que cada Raspberry Pi está subscripta a ese tópico y nos aseguraremos que cada ambiente controle sus artefactos de manera independiente. La convención que llevamos a cabo para implementar los mensajes del protocolo también es algo similar a una API

REST: **:type_of_gadget/:gadget_id/:action**

Donde **type_of_gadget** es el tipo de artefacto sobre el que se quiere actuar, ya sea **lights**, **tns** o **valves**, **gadget_id** es el identificador que comparten el servidor y el cliente, y **action** es la acción que se quiere llevar a cabo sobre el artefacto.

lights/:light_id/ON

Esta ruta sirve para encender una luz específica, y nos retornará un mensaje de éxito con el siguiente formato **rooms/:room_id/lights/:light_id/ON**, de lo contrario no responderá.

lights/:light_id/OFF

Esta ruta sirve para apagar una luz específica, y nos retornará un mensaje de éxito con el siguiente formato **rooms/:room_id/lights/:light_id/OFF**, de lo contrario no responderá.

valves/:valve_id/ON

Esta ruta sirve para abrir una canilla específica, y nos retornará un mensaje de éxito con el siguiente formato `rooms/:room_id/valves/:valve_id/ON`, de lo contrario no responderá.

valves/:valve_id/OFF

Esta ruta sirve para cerrar una canilla específica, y nos retornará un mensaje de éxito con el siguiente formato `rooms/:room_id/valves/:valve_id/OFF`, de lo contrario no responderá.

tv/:tv_id/ON

Esta ruta sirve para encender una televisión específica, y nos retornará un mensaje de éxito con el siguiente formato `rooms/:room_id/tvs/:tv_id/ON`, de lo contrario no responderá.

tv/:tv_id/OFF

Esta ruta sirve para apagar una televisión específica, y nos retornará un mensaje de éxito con el siguiente formato `rooms/:room_id/tvs/:tv_id/OFF`, de lo contrario no responderá.

Conclusión

Principalmente considero que el trabajo que llevamos a cabo fue exitoso y satisfactorio por varios motivos. En principio la idea de aplicar conceptos de las metodologías ágiles para llevar a cabo un proyecto con mucha incertidumbre fue efectivo. Se logró avanzar sin perder el ritmo, investigando para incorporar conceptos nuevos, siempre manteniendo la idea de hacer crecer la implementación iteración tras iteración, para finalmente lograr concluir en una aplicación realmente más grande de lo que se creyó en un principio.

El hecho de haber utilizado Raspberry Pi para implementar los clientes fue otro éxito. Realmente son unas computadoras muy potentes, sencillas y accesibles en cuanto a su costo. No sólo por las placas mismas, sino por todo el mundo de módulos que gira a su alrededor. Actualmente se están desarrollando cada vez más módulos nuevos, con distintas funcionalidades, que son interfaceables con las Raspberry Pi para lograr infinitudes de cosas. Además, todo el material que pudimos acceder en distintos sitios web para instruirnos y aprender sobre estos artefactos fue más que suficiente e importante.

Por otro lado fue muy satisfactorio lograr una implementación totalmente punta a punta. Es decir, los informáticos, generalmente sólo implementamos software, pero poder hacer algo de hardware es mucho más tangible y satisfactorio, realmente podemos ver las cosas andando y la sensación que provoca es totalmente distinta.

Finalmente nos quedamos con las ganas de seguir avanzando e investigando al respecto. Las aplicaciones posibles son infinitas y las cantidades de funcionalidades que se pueden abarcar son aún mayores. Los hogares migrarán hacia eso, tal como lo hacen las fábricas industriales, todo migra hacia la automatización. Esto nos permitirá tener mayor control sobre nuestros artefactos, brindando no sólo confort sino también eficiencia en cuanto al ahorro energético.



La tecnología está cada vez más al alcance de la mano, y el hecho de haber podido implementar IntelliHome manteniendo los costos accesibles nos da mucha mayor esperanza. Quedan muchos caminos y aspectos por investigar y explorar, pero nuevamente, las aplicaciones potenciales son infinitas.

Bibliografía

-  Python software foundation official website - www.python.org
-  Python - Wikipedia - es.wikipedia.org/wiki/Python
-  Raspberry Pi Foundation official website - www.raspberrypi.org
-  Foro de guías y tutoriales de Raspberry Pi - www.raspberrypi-spy.co.uk
-  Raspberry - Wikipedia - [es.wikipedia.org/wiki/Raspberry Pi](https://es.wikipedia.org/wiki/Raspberry_Pi)
-  Ruby official website - www.ruby-lang.org
-  Ruby - Wikipedia - es.wikipedia.org/wiki/Ruby
-  Ruby on Rails official website - www.rubyonrails.org
-  Ruby - Wikipedia - [en.wikipedia.org/wiki/Ruby on Rails](https://en.wikipedia.org/wiki/Ruby_on_Rails)
-  PostgreSQL official website - www.postgresql.org.es
-  Devise Github repository - github.com/plataformatec/devise