UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

SISTEMA DOMÓTICO HARDWARE Y SOFTWARE, BASADO EN UN SISTEMA EMBEBIDO Y UNA APLICACIÓN MÓVIL

Javier Rodríguez Inés
Tutor: Gonzalo Martínez Muñoz

MAYO 2017

SISTEMA DOMÓTICO HARDWARE Y SOFTWARE, BASADO EN UN SISTEMA EMBEBIDO Y UNA APLICACIÓN MÓVIL

AUTOR: Javier Rodríguez Inés
TUTOR: Gonzalo Martínez Muñoz

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Mayo de 2017

Resumen (castellano)

Este Trabajo de Fin de Grado consiste en construir un sistema software y hardware domótico que implemente una parte servidor en un sistema embebido que proporcionará distintos servicios y por otro lado una parte de aplicación móvil la cual consumirá dichos servicios.

El servicio principal a implementar consistirá en que la parte del servidor sea capaz de enviar una notificación al dispositivo móvil donde resida la aplicación cuando se realice una llamada en un portero automático, dando la posibilidad al usuario desde la aplicación móvil de abrir la puerta desde su dispositivo.

Estos servicios se implementarán dentro de un marco de transparencia de ubicación para el usuario, que podrá hacer uso de los mismos independientemente de su ubicación, siempre y cuando cuente con una conexión de red.

Además del servicio principal se implementaran otros servicios secundarios que serán atendidos por el servidor en el sistema embebido y serán consumidos desde la interfaz que implemente la aplicación móvil, facilitando de este modo su acceso al usuario. Estos servicios secundarios se basarán en utilidades básicas de un sistema domótico incluyendo nuevos servicios de accesibilidad y control domótico para el usuario, así como utilidades para el mantenimiento y monitorización del propio sistema.

La implementación hardware de la parte servidor se hará con una Raspberry Pi 3 la cual contará con Raspbian como sistema operativo el cual usará apache como servidor que alojará una Api en Ruby On Rails con Puma como servidor de aplicación y MySql como base de datos.

La parte de aplicación móvil será una aplicación Android que podrá ser usada en cualquier Smartphone Android con versión de Android 4.0 o superior, y hará uso de la conexión a internet del terminal.

A través de la aplicación Android el usuario podrá comunicarse con la api del servidor a través de peticiones HTTP de forma transparente que devolverán datos en formato Json que serán interpretados y visualizados en la aplicación móvil.

Abstract (English)

This Bachelor Thesis consist in develop a software and hardware domotic system implementing on one hand a server in a embed system with many services, and in the other hand a mobile app for consum that services.

The main function to implement in the server is send notifications to the mobile phone where the app is installed when a doorbell is ringed, giving the possibility of open the door to the app user.

The server services will be implemented behind a location transparency frame for the user, who could use this services regardless their location having always network connection.

In addition to the main service, other secondary services would be implemented to be attended by the embed system and use easily by the user from the interface implemented in the mobile app. The secondary services will be based on basics utilities of a domotic system for the user, with accessibility services and domotic control, maintenance tools and monitoring of the system.

Hardware implementation in the server side will be a Raspberry Pi 3 with Raspbian as operative system using also Apache as server to access an API in Ruby On Rails with Puma as application server and MySql for the database.

The app will be an Android app compatible with any Android smartphone with Android equal or over 4.0 version, and an operative internet conection.

With the Android app the user will be able to communicate with the API in the server throught HTTP petitions hidden for the user which return a Json format data that will be interpreted and show in mobile app

Palabras clave (castellano)

Domótica, internet de las cosas, sistema embebido, aplicación móvil, servidor, hardware, software, servicios, API , accesibilidad, automatización, portero automático, notificación

Keywords (inglés)

Home automation, internet of things, embedded system, mobile app, server, hardware, software, services, API, accesibility, automation, intercom, notification

Agradecimientos

Este trabajo de fin de grado nació de la idea de un pequeño proyecto de domótica con una única función sencilla que entrañaba una complejidad mayor por la infraestructura requerida para su funcionamiento. Esa idea inicial poco tiene que ver con este proyecto final, que poco a poco ha ido creciendo gracias a las sugerencias de algunas personas, gracias al apoyo de otras, gracias a la curiosidad de muchos, y gracias al interés de unos pocos.

Este proyecto no es solo mío, tiene un poco de todas esas personas que me han apoyado y ayudado a avanzar con el dándome valentía para enfrentarme a algo más grande, y haciéndome creer en que era posible desarrollar tal sistema pese a las limitaciones.

Este TFG no sería lo que es de no ser por todas esas personas, amigos, familiares, compañeros de trabajo, profesores y conocidos, muchos son los que han aportado un granito de arena que poco a poco ha hecho montaña, la montaña que es este proyecto, un proyecto con el que he disfrutado con tecnologías conocidas, y otras no tanto, yendo un poco más allá, investigando más y conociendo todos los recovecos, para finalmente acabar teniendo un gran proyecto multidisciplinar que trata software, hardware, programación, gestión de bases de datos y configuración y gestión de servidores.

Un proyecto empujado por mucha gente a la que tengo que agradecer que esto haya llegado a este punto. Muchas gracias a todas esas personas.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE ILUSTRACIONES

1 Introducción

1.1 Motivación

En los últimos años poco a poco ha ido apareciendo un término nuevo en el mundo de la tecnología, que lejos de quedarse atrás avanza cada día hacia horizontes menos convencionales para crear e innovar en cosas de lo más variadas, introduciéndose en campos en los que quizá hace unos años no nos habríamos imaginado que la tecnología podría cambiar de tal manera, y es que parece que la tecnología con internet como referente poco a poco se va adueñando de todo, de todas las cosas, es por eso que oímos hablar del *internet de las cosas* también conocido como IoT por sus siglas en inglés.

Ese término que hace referencia a cualquier objeto que pueda estar conectado a internet compartiendo información, facilitándonos su control o aportándonos datos.

Es un término bastante actual, pero fue Kevin Asthon en 1999 quien habló por primera vez de este concepto[1], y hoy en día hasta nuestras zapatillas están conectadas.

1.2 Objetivos

Como objetivo de este TFG se plantea crear una solución enmarcada dentro del paradigma del internet de las cosas para integrar varios dispositivos domésticos dentro de un mismo sistema de control permitiendo que estos estén interconectados y sean accesibles al usuario.

De este modo se busca aplicar soluciones del internet de las cosas a objetos cotidianos con la finalidad de facilitar el uso de los mismos y abrir un nuevo espectro de posibilidades dentro de estos objetos.

Básicamente esto es lo que plantea el internet de las cosas, una revolución tecnológica que trata por un lado de hacer más fácil la vida de las personas, a la vez que se dota a los objetos de conectividad que les permite alcanzar nuevas funciones o utilidades.

El proyecto busca cierta amplitud y escalabilidad siendo relativamente sencilla su extensión a nuevos objetos, pero se centrará en un caso práctico concreto, el cual se abordará como objetico principal, y será acercar el internet de las cosas a uno de los elementos más comunes en cualquier hogar, como es el alumbrado de una casa o el timbre o portero automático.

Así pues el objetivo principal será lograr conectar este elemento permitiendo recibir información e interactuar con el mismo.

1.3 Organización de la memoria

Esta memoria se va a organizar de acuerdo a los siguientes capítulos:

- Introducción: Capítulo introductorio que expondrá un contexto para ayudar a comprender el entorno, los objetivos y los resultados del proyecto, dando una imagen general del mismo así como preparando los términos que se introducirán en capítulos posteriores.

- Estado del Arte: Este capítulo tratará el contexto del proyecto exponiendo las condiciones técnico-culturales en el momento de su realización para dar una visión de la actualidad y los medios disponibles, permitiendo obtener un enfoque más amplio del problema y la tecnología de cara a afrontar el análisis del mismo.

- Diseño: Se desglosará el problema para exponer las distintas soluciones a considerar, así como las ventajas e inconvenientes de las mismas y la justificación de las elecciones tomadas.

- Desarrollo: Los detalles técnicos sobre el desarrollo y la implementación de la solución escogida durante el apartado anterior serán expuestos en esta sección con sus respectivas justificaciones y posibles alternativas dando un punto de vista más técnico de la solución y aportando distintas visiones de la misma.

- Integración pruebas y resultados: Este capítulo contiene la información relativa a la integración del proyecto así como las pruebas realizadas para verificar que cumple los requisitos propuestos, junto con las especificaciones y detalles de las mismas.

- Conclusiones y trabajo futuro: Por último se exponen las conclusiones obtenidas durante el proyecto junto con nuevas ideas surgidas durante el desarrollo que condicionarán el trabajo futuro.

2 Estado del arte

2.1 El Internet de las cosas

Este proyecto se encuadra dentro de una búsqueda de soluciones relacionadas con el paradigma del internet de las cosas, por lo que resulta interesante estudiar el estado del arte desde este punto de vista aportando algunas ideas sobre el IoT sus bases y su estado de desarrollo actual.

El IoT surge como una forma de llevar más allá la tecnología en nuestra vida diaria, una nueva forma de interconectar todo para hacernos la vida más sencilla

"Cuando hablamos de Internet de las Cosas nos referimos a un conjunto de ideas y pensamientos que ven una capa de conectividad digital en la parte superior de las infraestructuras y objetos existentes."[2]

La anterior cita expresa a la perfección desde un punto de vista físico el significado del IoT, ese conjunto de ideas que busca implantar la tecnología en objetos cotidianos, convertir cualquier pequeño objeto en algo conectado. Esto en realidad abre un mundo de posibilidades ya que tener cualquier objeto cotidiano conectado, permite un gran intercambio de información con estos objetos, es aquí donde entra en juego una nueva forma de interpretar la nube como medio de almacenamiento e interconexión con estos objetos, una plataforma de manejo de datos redefinida para adaptarse al internet de las cosas (Figura 1).

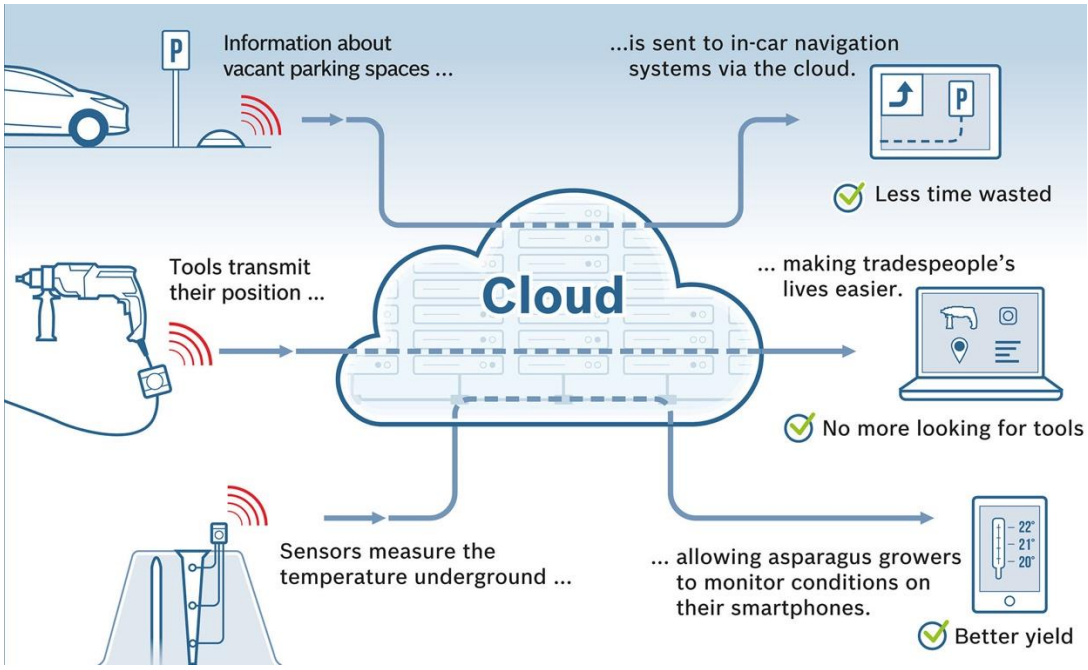


Figura 1: Representación del internet de las cosas [8]

Esta imagen representa de forma muy clara esta transformación digital y esa nueva forma de ver las cosas, una nube como medio centralizador y backend de dispositivos conectados que aparentemente no han cambiado su forma de funcionar ni necesitan una gran adaptación, simplemente proporcionan datos que serán transmitidos a la nube y serán procesados ahí, desencadenando las acciones pertinentes. Es aquí donde está la lógica, la mecánica de todo, en un medio aislado de los objetos que lo implementan, de modo que estos objetos simplemente tienen que conectarse, no precisan de ningún tipo de configuración o programación compleja simplemente dejan información en el lugar adecuado para ser tratada, son algo sencillo, un logger que simplemente registra datos, pero convertido en una potente herramienta gracias al procesamiento de estos datos.

En esto se basa el éxito del IoT, algo sencillo y liviano sin demasiadas complicaciones exprimido al máximo en un entorno remoto y transparente para el usuario.

2.2 Soluciones comerciales

Este proyecto busca crear un sistema de automatización de tareas domóticas accesible, escalable y personalizable, por ello no se centra únicamente en un campo de la domótica como algunas de las soluciones que se pueden encontrar en el mercado, por ello marca la diferencia respecto a otras soluciones comerciales que hay en la actualidad.

2.2.1 Tesa Entr Cerradura Inteligente

En el mercado podemos encontrar algunas soluciones hardware y software como la de Tesa Entr [3], que comercializan ya su cerradura inteligente que consiste en un bombín tradicional de cerradura que se coloca en la puerta quedando la cerradura tradicional hacia fuera y en el interior un pequeño sistema hardware a batería que es capaz de accionar la apertura o cierre de la puerta y con el cual nos podemos comunicar gracias a la conexión Bluetooth de un teléfono móvil, pudiendo así abrir la puerta de la forma tradicional con una llave o desde nuestro Smartphone.

2.2.2 Iomando

Existen algunos sistemas como Iomando[4], el cual ofrece algo parecido a algunos aspectos de este proyecto, permitiendo abrir puertas desde un teléfono móvil, consiste en una centralita que se debe instalar en el lugar donde se encuentra la puerta y controla la apertura de la misma, pudiendo accionarse este mecanismo desde uno o varios terminales móviles, cuenta también con una aplicación móvil la cual es necesaria para conectarse a dicha centralita. En cuanto a opciones de interfaz se ofrece un panel de control web suministrado por la centralita en el cual se pueden configurar algunos parámetros como las horas a las que se desea restringir la apertura de la puerta o los usuarios que tienen acceso a la misma.

Iomando dispone también de una API con la que ofrece adaptación e integración de su solución con otras aplicaciones externas. Disponen de aplicación Android e IOS. Esta solución está basada en software privativo y es de pago, se paga por su instalación y uso así como la por la integración con otras aplicaciones. Esta solución está limitada únicamente al control de accesos y no ofrece soporte para otras tareas domóticas.

2.2.3 Parkingdoor

Parkingdoor[5] es un sistema similar al anterior, pero que se centra en la apertura desde el móvil de puertas de garaje concretamente, ofreciendo una centralita que se comunica con los receptores de las puertas de garaje automáticas permitiendo usar nuestro smartphone como si fuera el mando del garaje. Se centra en un nicho de mercado muy concreto destinado a la compartición y alquiler de plazas de garaje privadas, pudiendo permitir a cualquier persona la apertura de la puerta con su móvil y revocar su acceso una vez concluido el alquiler.
Una vez más es una solución muy concreta que no abarca otras tareas domóticas

2.2.4 Bombillas Wifi

Últimamente en el mercado han aparecido productos se controlan independientemente desde el móvil, vía Wifi, son bastantes populares en este campo las bombillas. Phillips fue una de las primeras marcas en comercializar este tipo de productos, con sus bombillas Phillips Hue[6], las cuales se pueden controlar desde el móvil, encendiéndolas y apagándolas, o cambiando su intensidad o color.

2.2.5 Cámaras Wifi

Muchas son las cámaras IP que podemos encontrar en el mercado que nos permiten conectarlas a la red mediante cable o bien por Wifi y poder acceder a sus imágenes a través de una URL concreta.

3 Diseño

3.1 Análisis del problema

El problema a abordar consiste en la construcción de un sistema para gestionar aspectos del IoT dentro de un entorno domótico, pudiendo controlar distintos dispositivos de manera centralizada en este sistema.

La solución más adecuada a dicho problema pasa por implementar un sistema hardware y software que comprenda principalmente dos elementos básicos, una parte de gestión y centralización de los dispositivos conectados, y otra parte de interfaz accesible para el usuario.

Como hemos podido observar en el análisis del estado del arte, casi todas las soluciones similares tienen un denominador común, que es el elemento centralizador de todos los componentes. En la mayoría de proyectos este elemento toma el nombre de centralita, siendo el nexo de unión entre los dispositivos del internet de las cosas y el usuario, quedando como nodo central de esa unión el procesamiento de los datos y lógica necesaria para el control de los dispositivos.

En adelante se hará referencia al elemento hardware central del sistema como centralita, a la cual se conectarán distintos elementos a los cuales nos referiremos como dispositivos IoT y que separaremos en dos grupos, dispositivos IoT cableados y dispositivos IoT inalámbricos.

En cuanto al aspecto de la conexión es posible que lo más eficiente sea usar únicamente dispositivos IoT inalámbricos ya que podemos descentralizarlos del resto del proyecto dotándolos de cierta movilidad e independencia, pero también hay escenarios en los cuales los dispositivos IoT cableados pueden ser más eficientes o incluso la única posibilidad, por ejemplo si necesitamos cumplir ciertos objetivos de rendimiento o disponibilidad, una conexión cableada nos aportará mayor velocidad y disponibilidad, o incluso se puede dar el caso de que no dispongamos de una conexión inalámbrica entre los elementos. Por lo tanto en este proyecto se implementarán ambos tipos de dispositivos para cubrir todo el espectro de alternativas

3.2 Diseño de la solución

Para el elemento hardware la solución más acertada sería un sistema embebido con un pequeño servidor a modo de nodo centralizador de las conexiones con el resto de objetos o dispositivos del sistema, ya que como se ha comentado en el apartado anterior esta parte será la encargada de centralizar la lógica de negocio y el procesamiento necesario para garantizar el correcto funcionamiento de todos los dispositivos.

Como se precisa implementar una solución escalable, es necesario que este elemento sea flexible a la hora de realizar posibles evolutivos, o ampliar la capacidad del sistema.Con estas reflexiones aparecen algunos posibles candidatos para la implementación dentro del conjunto de sistemas embebidos de hardware libre de bajo coste, como podrían ser Arduino, Raspberry, Chip u Orange Pi.

En cuanto al elemento software es casi obligado que este se trate de una aplicación móvil capaz de conectarse al sistema hardware. Esta alternativa es la que mayor flexibilidad ofrece junto con una mejor experiencia de usuario.
Otras alternativas como dispositivos de control remoto o interfaces graficas implementadas en la propia centralita quedan rápidamente descartadas por algunos inconvenientes como la falta de movilidad, la poca flexibilidad de cara a la escalabilidad del proyecto o su coste de producción.

Mientras que una aplicación móvil nos aporta movilidad, control sencillo y siempre accesible, reducido coste de implementación, facilidades para la actualización y mejora, fácil distribución, y mejor experiencia de usuario entre otras ventajas.

3.3 Formalización del diseño: Arquitectura de la solución

A continuación se da una visión formalizada de la arquitectura de la solución del proyecto sentando las bases de la disposición de los elementos así como su intercomunicación.

Para comenzar, el esquema de la Figura 2 ayudará a comprender las interconexiones de todos los elementos del sistema

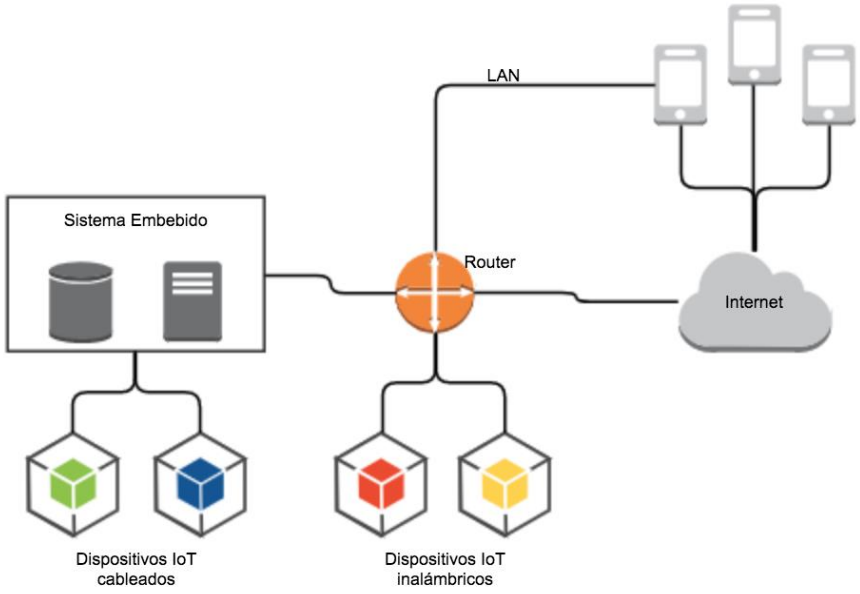


Figura 2 Esquema de conexiones entre elementos del sistema.

3.3.1 Centralita

3.3.1.1 Hardware

La centralita es el sistema embebido mostrado en la Figura 2 como sistema embebido, y está conformado por una Raspberry Pi 3 Modelo B. La justificación de esta decisión de diseño

viene dada por la versatilidad que aporta dicho sistema, y su gran flexibilidad, siendo posible instalar en el mismo una gran variedad de distribuciones Linux de muy variadas características que podrán alojar un servidor, así como otros sistemas operativos como Windows 10 IoT diseñado especialmente para este tipo de proyectos.

La Raspberry Pi 3 Modelo B aporta las siguientes especificaciones:

CPU de 4 núcleos a 1.2GHz de 64-bit ARMv8
1GB de RAM
802.11n Wireless LAN
Bluetooth 4.1
Bluetooth de bajo consumo (BLE)
4 puertos USB
40 pines GPIO
Puerto Full HDMI
Puerto Ethernet
Jack de 3.5mm combinado de audio y video compuesto
Interfaz de cámara (CSI)
Interfaz de display (DSI)
Slot para tarjeta Micro SD
VideoCore IV 3D núcleo grafico

Sus cuatro núcleos nos asegurarán un buen rendimiento y disponibilidad combinados con la implementación de un servidor multihilo capaz de atender peticiones concurrentes a nuestra centralita.

Está dotada de un gran número de alternativas de conectividad, las cuales podremos aprovechar para el desarrollo del proyecto, como el Wifi y Bluetooth para las conexiones, así como el puerto Ethernet si no disponemos de una conexión inalámbrica o preferimos usar una cableada, los puertos de video HDMI y de video compuesto nos permitirán interaccionar con la Raspberry de forma sencilla gracias a interfaces gráficas, pudiendo configurar parámetros o incluso desarrollar fácilmente desde el propio dispositivo.

Los puertos USB nos permiten la conexión de múltiples periféricos pudiendo hacer uso de cámaras web o similares de forma sencilla.

La interfaz GPIO compuesta por 40 pines nos permitirá la comunicación sencilla y rápida con dispositivos digitales de forma cableada. Un punto en contra en este aspecto es que la Raspberry no dispone de conexiones analógicas en su GPIO por lo que deberemos convertir este tipo de señales si queremos hacer uso de ellas.

Disponemos también de interfaces de cámara y de display pudiendo hacer uso de las cámaras oficiales de Raspberry que ofrecen buenas resoluciones, incluso cámaras de visón nocturna. En cuanto al display, podremos conectar una pantalla táctil directamente a esta interfaz para interactuar o configurar ciertos parámetros de forma rápida y sencilla.

En resumen la Raspberry Pi 3 Modelo B nos aporta una gran potencia de procesamiento que nos permitirá escalar nuestro proyecto sin necesidad de restructurar el hardware, y cumplir con los requisitos de disponibilidad y rendimiento. También nos aporta un gran número de posibilidades de conexión tanto cableadas como inalámbricas

Arduino por su parte queda descartado por su baja capacidad de computo que no nos permitirá mantener ciertos requisitos de rendimiento si tenemos un número alto de usuarios y operaciones complejas en nuestra centralita, además de ofrecernos una conectividad básica muy limitada que se debería ampliar con nuevos componentes que encarecerían el coste total y complicarían el desarrollo. Tampoco disponemos de una interfaz gráfica desde donde desarrollar o configurar el dispositivo como en el caso de la Raspberry. Por otro lado como ventaja aportaría entradas y salidas analógicas en su GPIO de lo cual carece la Raspberry.

Otras soluciones similares a la Raspberry como Orange Pi nos aportan unas especificaciones muy similares pero con un menor "soporte" por ser plataformas menos extendidas. Y no suponen demasiada diferencia en cuanto a prestaciones por lo tanto usaremos Raspberry por tener una comunidad de usuarios mayor que nos ayudará en la resolución de problemas básicos y a la hora de encontrar sistemas operativos adaptados a Raspberry y librerías diseñadas específicamente para esta plataforma.

3.3.1.2 Software

La Raspberry actuará como centralita gracias a la implementación de una API basada en el protocolo HTTP que será consumida tanto por los dispositivos IoT conectados a ella como por los usuarios a través de la aplicación móvil.

Esta API será implementada en lenguaje Ruby, que se trata de un lenguaje interpretado no tipado y de sintaxis sencilla que permite un desarrollo rápido y eficiente, facilitando el testeo de la aplicación. Complementariamente se utilizará el framework Rails, el cual aporta grandes ventajas a la hora de desarrollos web.

Rails agiliza el trabajo proporcionando un sistema de rutas fácilmente configurable y escalable, generación de código automática para agilizar el desarrollo y una robusta infraestructura de modelo vista controlador de base que nos ahorrará tiempo de desarrollo y restará complejidad al código.

Rails también ofrece protección contra "Cross-site request forgery", un tipo de exploit malicioso capaz extraer información y realizar acciones potencialmente peligrosas en webs vulnerables a dicho tipo de ataque

El formato API se ha escogido por ser lo más conveniente para este tipo de proyecto aportando ligereza a las peticiones, por no sobrecargar el servidor y resultar muy sencillo de interconectar con el resto de componentes gracias al uso de peticiones HTTP. Dejando como opción futura la creación de vistas para interactuar con la API a través de un navegador, lo cual sería muy sencillo gracias al uso de la tecnología Ruby On Rails que se ha elegido, ya que gracias al framework Rails se pueden generar automáticamente vistas para interactuar con los modelos en formato de aplicación CRUD (create, read, update, destroy) o interactuar con los controladores

3.3.2 Dispositivos IoT

En esta sección daremos una visión del diseño a seguir para implementar dispositivos IoT, los cuales estarán conectados a la centralita, y según su forma de conectarse a esta distinguiremos dos tipos:

3.3.2.1 Dispositivos IoT cableados

Estos dispositivos se conectarán directamente a la Raspberry a través de cables usando las distintas interfaces y puertos que nos ofrece.

Por lo general las conexiones se realizarán por USB para dispositivos estandarizados que dispongan de este tipo de conexión, o a través de la interfaz GPIO de la Raspberry para dispositivos que no dispongan de una conexión USB.

El GPIO de la Raspberry admite únicamente conexiones digitales por lo que no podremos hacer uso de señales analógicas sin previa conversión de las mismas, pero para evitar esto haremos uso únicamente de dispositivos digitales. También cabe destacar que la alimentación proporcionada por el GPIO es de 5 voltios por lo que usaremos dispositivos que trabajen con este voltaje.

En el proyecto usaremos dos dispositivos IoT cableados, serán el sensor de temperatura y humedad DHT-11 y una cámara web Logitech C110

3.3.2.1.1 DHT-11 Sensor de temperatura y humedad digital

El DHT-11 es un sensor de temperatura y humedad bastante versátil puesto que incorpora los dos tipos de sensores en uno permitiendo una fácil lectura de ambos valores.

Este sensor es totalmente digital por lo que podremos leerlo sin problemas y sin necesidad de ningún tipo de conversión desde el GPIO de la Raspberry. Los datos son devueltos con señales digitales (cero o uno) en forma de bits, el sensor devuelve una cadena de 40 bits la cual se divide en 2 bytes de temperatura, 2 bytes de humedad y un byte de paridad. Esta y otras especificaciones del componente las podemos encontrar en su datasheet original. [9]

En cuanto al voltaje, el DHT-11 es capaz de trabajar con 3,3v o con 5v indistintamente por lo que no supondrá ningún inconveniente.

3.3.2.1.2 Logitech C110 cámara web USB

Este modelo de cámara web cableada dispone de conexión USB por lo que nos permitirá conectarla fácilmente a la Raspberry en cualquiera de sus 4 puertos USB.

Su resolución óptica es de 640x480 lo cual será suficiente para este proyecto ya que no precisamos de una resolución mayor, pero en caso de querer obtener imágenes de mayor

resolución disponemos de un amplio catálogo de cámaras web USB que podremos usar sin mayor problema.

3.3.2.2 Dispositivos IoT inalámbricos

Por otro lado tendremos una serie de dispositivos IoT conectados sin cables para estudiar también las características e implementación de este tipo de dispositivos en el proyecto.

Las formas de conexión disponibles que tenemos en nuestra centralita son las que nos brinda la Raspberry, entre las que contamos con Wifi y Bluetooth.

En esta ocasión haremos uso únicamente de la conexión Wifi para comunicarnos con los dispositivos.

En este proyecto el dispositivo conectado de forma inalámbrica será el timbre o video portero, que no necesitará cables para conectarse a la centralita.

3.3.2.2.1 ESP8266 ESP-01

El ESP8266 es un pequeño microcontrolador de bajo coste que incorpora un procesador RISC de 32 bits y 80MHz (con posibilidad de hacerle overclock a 160 MHz) y conectividad Wifi y GPIO.

El ESP-01 (Ilustración 3) es la versión más extendida del ESP8266 y se presenta en forma de un pequeño chip con 8 pines de conexión que se alimenta con 3,3 voltios.

El ESP8266 tiene una arquitectura Harvard con 64KB de RAM para instrucciones y 96KB para datos.

Este pequeño microcontrolador será el que nos permita dotar de conectividad y capacidad de procesamiento a cualquier dispositivo que queramos incorporar a nuestra red IoT. Simplemente debemos conectarlo al dispositivo y cargar en este microcontrolador el código para realizar las funciones necesarias, como por ejemplo leer datos del dispositivo conectado y enviarlos a la centralita.

3.3.3 Aplicación Móvil

La aplicación móvil será la parte que permita interactuar al usuario con el resto del sistema y los dispositivos conectados a este, siendo el principal medio de control del sistema.

La aplicación de desarrollará para dispositivos Android ya que es el sistema operativo más usado del mundo por encima incluso de Windows, y en el terreno de los móviles su uso es claramente superior al de iOS, según datos de StatCounter [10] como podemos observar en la Figura 3

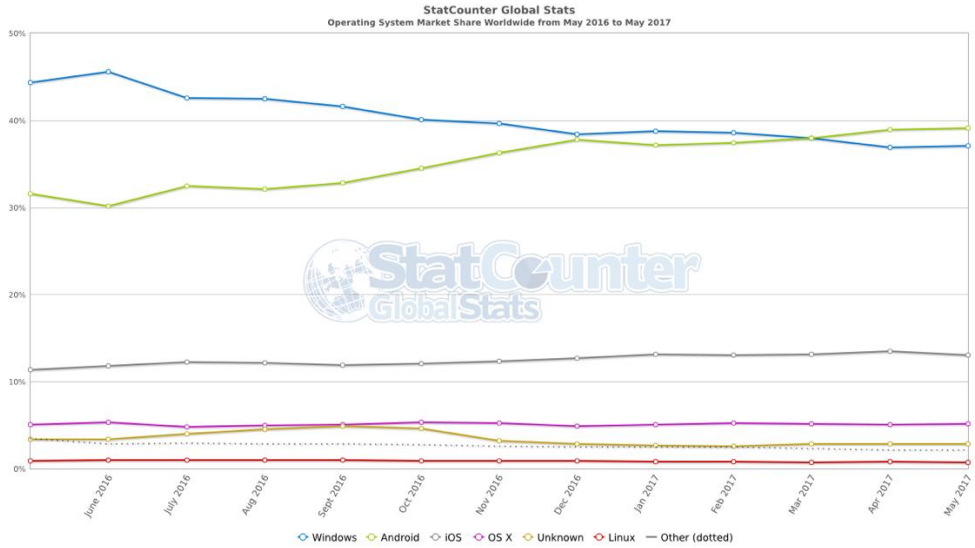


Figura 3 Comparativa uso de sistemas operativos a nivel global - StatCounter (http://gs.statcounter.com/os-market-share)

Para desarrollar la aplicación móvil para dispositivos Android el lenguaje tendrá que ser Java, como entorno de desarrollo se usará Android Studio y se simulará un terminal virtual con Android puro para realizar las pruebas.

La comunicación entre la Raspberry y la aplicación Android se hará a través de peticiones HTTP, para lo cual la opción más sencilla es el uso de la biblioteca Volley de Google[11].

3.4 Requisitos

3.4.1 Requisitos funcionales

A continuación se muestran los requisitos funcionales del sistema:

- El sistema controlará tareas de tipo domótico aplicando la filosofía IoT.
- Las distintas tareas serán previamente programadas y en ciertos casos podrán ser configurables de acuerdo a ciertos parámetros preestablecidos.
- El usuario debe poder conectarse a través de una aplicación móvil al sistema para interactuar con él.
- El usuario obtendrá un aviso en su dispositivo móvil cuando alguien llamé al portero automático donde esté instalado el sistema
- Cuando se produzca el evento anterior el usuario podrá abrir la puerta desde su dispositivo móvil

- El usuario podrá activar y desactivar otros dispositivos a través del sistema domótico como luces y aparatos electrónicos.

- El usuario podrá recibir imágenes a través de una cámara instalada en el sistema.

- El usuario podrá obtener datos de sensores de temperatura y humedad conectados al sistema.

3.4.2 Requisitos de interfaz y usabilidad

A continuación se muestran los requisitos de usabilidad:

- El sistema debe contar con una interfaz gráfica con la que pueda interactuar el usuario para configurar y hacer uso del sistema.

- La interfaz debe ser sencilla y visual, y se presentará como una aplicación móvil.

3.4.3 Requisitos de documentación

A continuación se muestran los requisitos de documentación:

- El proyecto debe contar con documentación respecto a la programación de nuevas funcionalidades.

- El código desarrollado debe estar documentando de forma que sea fácil diseñar nuevos evolutivos y comprender los algoritmos implementados

3.4.4 Requisitos de mantenibilidad y portabilidad

A continuación se muestran los requisitos de mantenibilidad y portabilidad:

- El sistema debe ser portable pudiendo instalarse en cualquier lugar con independencia de las condiciones del mismo salvo la necesidad de conexión a internet y alimentación para el mismo

- El sistema debe permitir su uso de forma remota desde cualquier lugar.

- El sistema debe ser configurable de forma remota una vez que se encuentre instalado.

- El sistema debe ser fácilmente actualizable por un usuario cualificado, así como permitir mejoras y evolutivos sobre el código.

- El sistema se debe desarrollar de manera escalable y mantenible.

3.4.5 Requisitos de seguridad

A continuación se muestran los requisitos de seguridad:

- El sistema debe proveer mecanismos de seguridad para evitar el acceso no autorizado a la información o funciones del mismo.

- La autenticación de los usuarios se hará de forma segura y confiable para el usuario.

3.4.6 Requisitos operacionales

A continuación se muestran los requisitos operacionales:

- El sistema debe contar con mecanismos de recuperación en caso de desconexión o cuelgue de cualquier parte del mismo.

- No serán necesarios mecanismos de back-up, pero se desarrollará de forma escalable en ese sentido.

3.5 Usuarios y tareas

El sistema contará con un único tipo de usuario que será el que tenga acceso a toda la funcionalidad del mismo, debiendo emparejar sus dispositivos

3.6 Representación

Maquetas de la aplicación en el Anexo C

4 Desarrollo

4.1 Sistema embebido

Para comenzar con el desarrollo de la parte del sistema embebido tomaremos conciencia de todos los elementos de este y de su disposición en el sistema para a continuación pasar a describirlos.

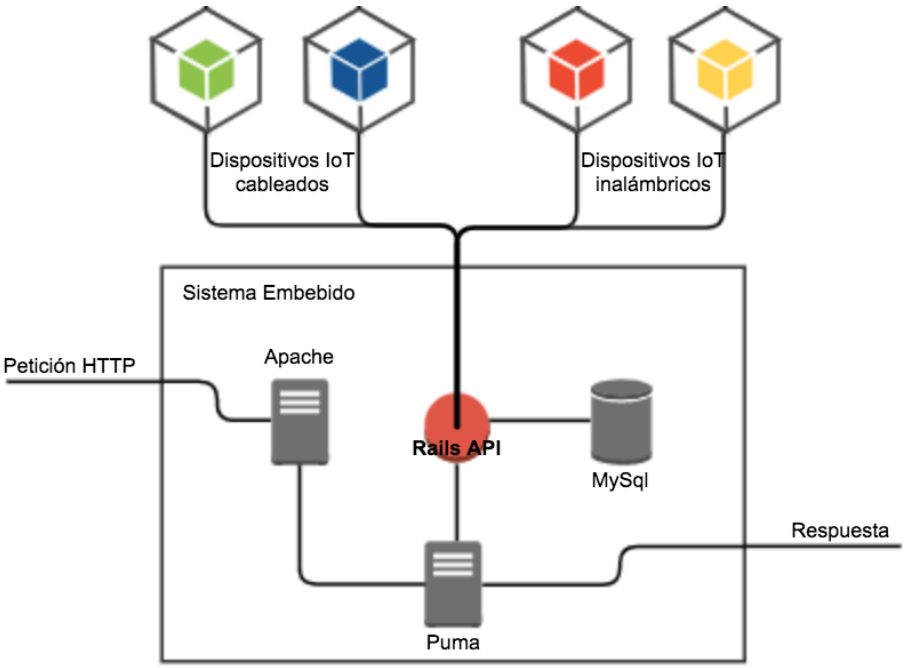


Figura 4 Representación del sistema embebido

Como vemos en la representación (Figura 4), el sistema embebido está formado por varios elementos, que son el servidor web, se utilizará Apache, el servidor de aplicación, que será Puma, la base de datos, MySql, y la aplicación que será la API implementada en Ruby On Rails.

4.1.1 Raspberry, primeros pasos

Lo primero que debemos configurar en nuestro sistema embebido será la Raspberry, la cual necesitará una tarjeta micro SD en la que hayamos cargado previamente un sistema operativo.

La Raspberry soporta varios sistemas operativos que van desde multitud de distribuciones Linux como Ubuntu Mate o Raspbian, a Windows IoT, una distribución de Windows pensada para el IoT, pasando por sistemas más específicos para usar la Raspberry como centro multimedia, como por ejemplo OSCM.

El sistema que mejor encaja con la Raspberry y más libertad nos dará en este proyecto es Raspbian, una distribución Linux muy manejable que podremos instalar fácilmente en una tarjeta micro SD, a partir de la imagen del sistema que podemos descargar de la web oficial de Raspberry:

https://www.raspberrypi.org/downloads/raspbian/

Una vez descargada la imagen podremos instalarla en la tarjeta micro SD con multitud de aplicaciones, en este caso se ha utilizado Apple Pi Baker para Mac (Figura 5), que permite a través de una sencilla interfaz elegir la imagen e instalarla en la SD.

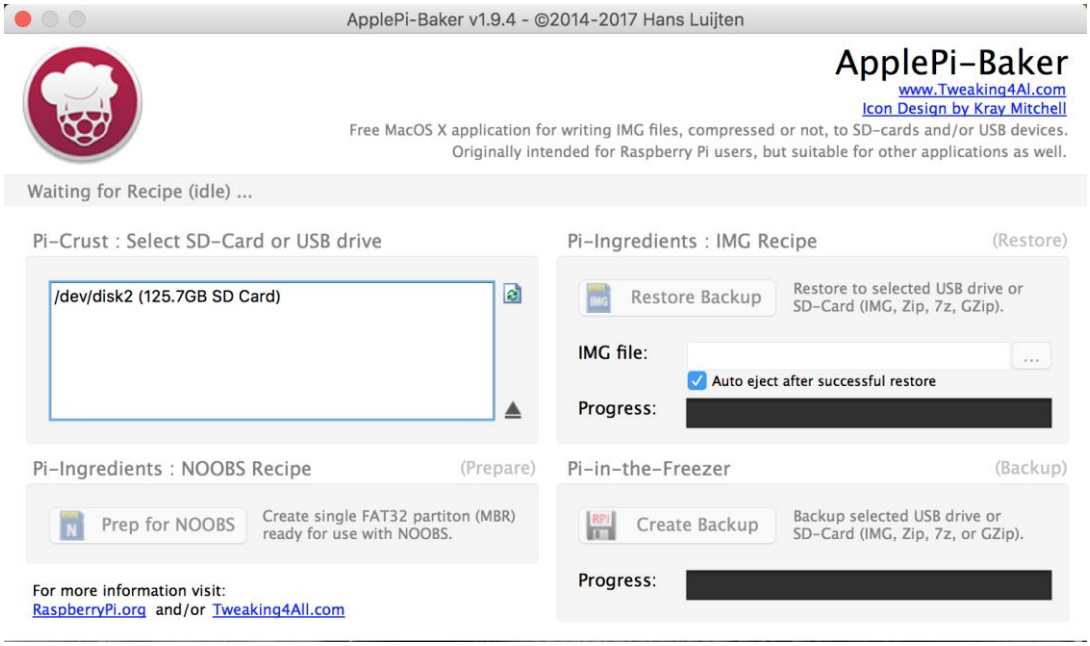


Figura 5 Interfaz de la aplicación Apple Pi Baker

Una vez que tengamos la tarjeta micro SD preparada con la imagen del sistema operativo tan solo debemos insertarla en la Raspberry y conectarla a la corriente a través de un cable micro USB que le proporcione 5 voltios y una intensidad de 2,5 Amperios.

A partir de este punto es posible utilizar el conector HDMI de la Raspberry para conectar un monitor y proseguir la configuración a través de la interfaz gráfica que provee Raspbian(Figura 6).

Entre los primeros pasos de configuración esta ejecutar el comando `raspi-config` que provee Raspbian y nos ofrecerá una serie de apartados básicos a configurar donde podremos habilitar la conexión ssh o el GPIO entre otros aspectos.

Tras eso es conveniente ejecutar `sudo apt-get update` para actualizar los paquetes antes de instalar software desactualizado.

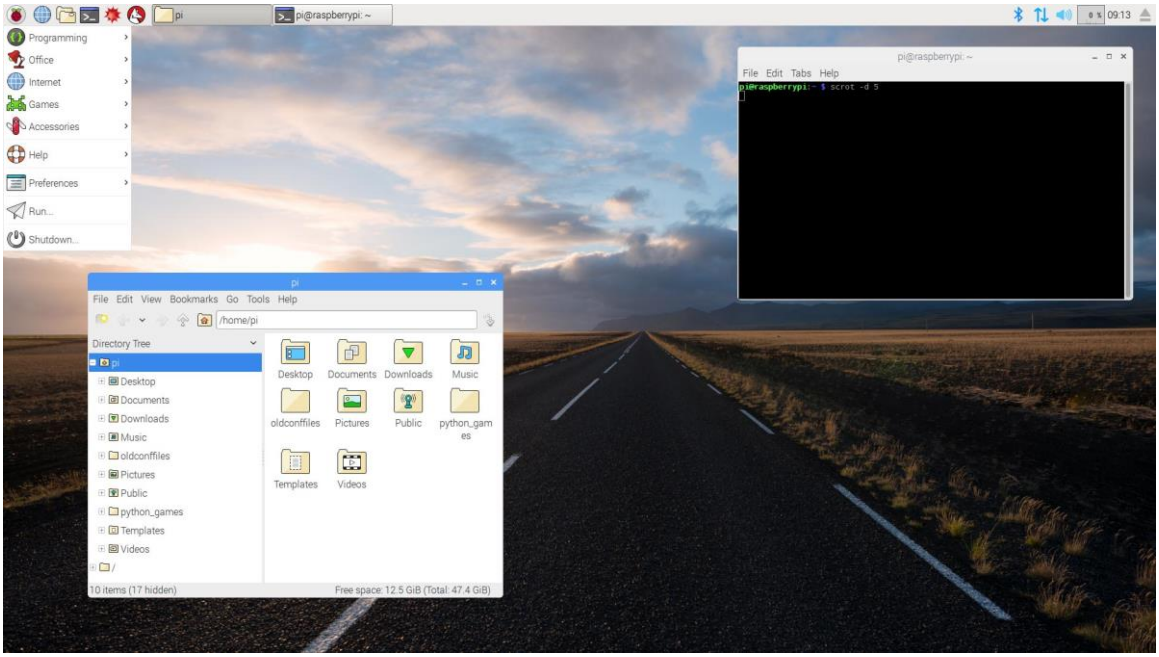


Figura 6 Interfaz gráfica de Raspbian

4.1.2 RVM

Será importante instalar Ruby y el framework Rails para poder ejecutar la aplicación. Esto se puede hacer manualmente pero existe una alternativa interesante que nos ayuda a controlar de forma eficiente las versiones de Ruby y Rails, esta alternativa es RVM (Ruby Version Manager)[12]

RVM se instala de forma conjunta instalando en la maquina Ruby junto con Rails si así se desea, y permite un rápido control y actualización de las versiones de Ruby, permitiéndonos incluso tener distintas versiones instaladas y cambiar entre ellas de forma rápida.

RVM además de facilitarnos la instalación nos será de gran ayuda si en el futuro decidimos alojar otra aplicación conjuntamente en la Raspberry o si en algún momento actualizamos la versión de Ruby o de Rails de nuestra aplicación, por lo tanto este proyecto utilizará RVM contribuyendo así a la escalabilidad del mismo.

Para la instalación deberemos seguir las instrucciones que encontramos en el apartado de instalación de la web oficial de RVM[12]

El primer paso es instalar las claves GPG de verificación de los paquetes. Si se ha instalado RVM en varias ocasiones es fácil saber que el servidor de claves proporcionado por la web suele dar errores y timeout fácilmente, la solución a este problema es esperar a que el servidor vuelva a estar operativo o buscar un servidor alternativo en la red.

A continuación la web nos ofrece varias alternativas de instalación mediante comandos curl, que van desde la instalación de Ruby únicamente hasta instalaciones más completas con Ruby, Rails y Puma por ejemplo

Tras ejecutar el curl correspondiente a nuestras necesidades el instalador de RVM lo hará todo.

4.1.3 Bundler

Bundler es la herramienta que nos permite instalar fácilmente las gemas de nuestra aplicación y mantenerlas actualizadas evitando conflictos con sus dependencias.

En Ruby las gemas son las librerías, que aportan nuevas funciones y encapsulan código. No son otra cosa más que las librerías que todo lenguaje tiene pero con un nombre distinto. Si es destacable la gran comunidad de desarrolladores que tiene Ruby y la cantidad de gemas, prácticamente para todo, que podemos encontrar gracias a ellos.

Una aplicación Rails contiene en su raíz un fichero Gemfile, el cual es un listado de todas las gemas que utilizará la aplicación.

Bundler es una gema que tenemos que instalar con el comando `gem install bundler`, y una vez que hayamos hecho esto no volveremos a instalar manualmente ninguna gema, ya que con el comando `bundle install` se instalarán automáticamente todas las gemas del Gemfile y sus dependencias.

4.1.4 Apache

El servidor web se implementa con apache2, concretamente la versión Apache/2.4.10 la cual se instala de forma sencilla desde línea de comandos

```
sudo apt-get install apache2
```

La configuración de apache es el punto más importante del servidor web, se debe configurar correctamente para que las peticiones recibidas sean correctamente dirigidas al servidor de aplicación.
Apache concentra su configuración en el directorio /etc/apache2, donde podemos encontrar varios subdirectorios y archivos con configuración.

Apache es capaz de servir distintas web o aplicaciones y gestiona su configuración por "sites" (Figura 7), de este modo encontramos dos directorios importantes que son "sites-available" y "sites-enabled" el primero es el que contiene los archivos de configuración, típicamente uno por sito, de cada recurso (sitio, aplicación, etc.) que gestiona apache, de este modo que contiene la configuración disponible, mientras que la configuración habilitada se encuentra en el "sites-enabled", siendo típicamente los archivos de esta carpeta enlaces simbólicos a archivos de "sites-available"

De modo que para habilitar la aplicación se ha escrito un archivo de configuración para la misma, se ha colocado en la carpeta "sites-available", y se ha hecho un enlace simbólico al mismo desde la carpeta "sites-enabled", esto último se realiza fácilmente gracias al comando "a2ensite" de Apache.

El archivo o site utilizado en este caso es el siguiente:

```
[pi@raspberrypi:~ $ cat /etc/apache2/sites-enabled/rasp_api.conf
<VirtualHost *:80>
  # ServerName www.rasp_api.es
  #ServerAlias rasp_api.es
  DocumentRoot /var/www/rasp_api/current/public

  <Location /assets>
    ProxyPass !
  </Location>
  <Location /system>
    ProxyPass !
  </Location>

  ProxyPass / http://127.0.0.1:3000/
  ProxyPassReverse / http://127.0.0.1:3000/
</VirtualHost>
```

Figura 7 Site de apache

Como podemos observar en la imagen, en el site se define un virtualhost que escuchará peticiones en el puerto 80.

Podemos definir un ServerName y un ServerAlias en caso de que tuviéramos un nombre de dominio para nuestro sitio.

El siguiente punto importante es el DocumentRoot, que debe ser el directorio public de nuestra aplicación Rails.

Y por último, configuramos el ProxyPass y ProxyPassReverse para hacer las redirecciones al puerto 3000 que será donde tengamos corriendo la aplicación Rails gracias al servidor de aplicación Puma.

4.1.5 Puma

Puma es uno de los servidores de aplicación disponibles para Rails con un manejo muy efectivo de la memoria y una configuración muy optimizable gracias al uso de workers e hilos, los cuales deberemos configurar correctamente junto con el pool de conexiones de la base de datos para ofrecer el mejor servicio posible.

Puma se implementa con Rails gracias a su propia gema y tan solo necesitaremos un pequeño archivo de configuración en el servidor para hacer que este servidor de aplicación funcione junto con Apache.

El funcionamiento de ambos servidores juntos se configura designando un puerto de despliegue para Puma, o un socket tcp y configurando el site de Apache mediante redirección proxy a dicho puerto.

4.1.6 MySql

Para la instalación de MySql en la Raspberry, deberemos instalar el servidor de MySql, lo cual se hace mediante el siguiente comando en sistemas Linux:

```
sudo apt-get install mysql-server
```

Tras ejecutar dicho comando la instalación comenzará y tan solo deberemos seguir los pasos que se indiquen durante la misma, ya que se nos pedirá introducir una contraseña para el usuario root de la base de datos.

Un vez instalado MySql podremos administrar la base de datos remotamente conectándonos con el usuario y la contraseña creados a la IP de la Raspberry en el puerto que se encuentre la base de datos.

En este proyecto la base de datos no será muy significativa ya que solo se guardará en ella el token de autenticación de cada usuario.

La base de datos únicamente contendrá una tabla de usuarios que guardará el token de autenticación que se requerirá en cada petición.

4.1.7 Rails app

Rails es un framework web para la creación de aplicaciones sobre lenguaje Ruby que agiliza la codificación de aplicaciones web aportando una infraestructura básica y gran cantidad de generadores de código.

El comando `rails new` es suficiente para empezar una aplicación Rails que creará toda la estructura de directorios, y componentes básicos de la aplicación.

4.1.7.1 MVC

Las aplicaciones Ruby On Rails están basadas en el paradigma modelo vista controlador, una visión que separa en tres componentes principales el código de una aplicación para crear una estructura limpia sólida y bien organizada que permita la escalabilidad y mantenimiento de la aplicación a la vez que mejora el rendimiento de la misma.

El modelo es la parte que trata la estructura de datos, modelizándola conseguimos lograr un nivel de abstracción y encapsulamiento. En Rails esta parte está es gestionada por ActiveRecord, que maneja la persistencia de nuestra aplicación y nos proporciona una sintaxis de alto nivel para interactuar con la base de datos.

Tan solo debemos usar unos sencillos comandos para generar los modelos de la aplicación y ActiveRecord se encargará de crear las migraciones para crear las tablas en base de datos y de mantener la integridad de la misma con sus relaciones.

Lo más óptimo es tratar de centralizar en esta parte toda la lógica de la aplicación relativa al modelo de datos que será la parte más pesada debido a las conexiones con la base de datos.

En cuanto a la parte del controlador, es la que actúa de intermediaria entre la vista y el modelo, tiene cierta lógica orientada a la vista y al consumo de los datos, y es aquí donde se debe implementar la lógica de negocio de la aplicación no relativa a modelo.

Por último la vista es la parte que se consume del lado del cliente, y por ello debe ser lo más ligera posible evitando cualquier tipo de lógica en ella y por supuesto prescindiendo de todo tipo de conexión con la base de datos.

La vista esta ideada para recoger los datos procesados en el controlador y representarlos al usuario permitiéndole interaccionar con el modelo de datos a través del controlador.

4.1.7.2 Autenticación y seguridad

La aplicación contará con un sistema de autenticación por token, que será requerido en cada petición. Se desarrollará un controlador de usuarios, junto con su propia ruta protegida de modo que esta solo sea accesible desde la misma red en la que se encuentra la Raspberry. El controlador de usuarios recibirá la petición de la aplicación móvil para emparejar un dispositivo móvil, y simplemente generará un token que guardará en base de datos y responderá a la petición con dicho token. La aplicación móvil guardará ese token y lo usará en todas las peticiones futuras.

Todas las peticiones que se hagan a la Raspberry deben ir acompañadas del token de autentificación que debe ser válido (se comprobará que está en base de datos). La única ruta accesible sin token será la de registrar nuevo usuario, pero dicha ruta solo será accesible desde la misma red local, no desde fuera, es decir la aplicación móvil para registrarse en la Raspberry debe estar conectada a la misma red Wifi que esta.

```
Rails.application.routes.draw do
  constraints(ip: /192\.\d+\.\d+\.\d+/) do
    resources :users
  end
```

Figura 8 Capado de rutas por IP

Para conseguir esto caparemos la ruta con una restricción por IP aplicando una expresión regular para aceptar solo accesos a esa ruta si la IP es local (por ejemplo si empieza por 192.) como se puede observar en la Figura 8

El registro de los usuarios se realiza en el users_controller (Figura 9) generando aleatoriamente un token. El campo nombre se ha creado dentro del modelo User, y aunque no tiene funcionalidad actualmente, sería interesante obtener el nombre del dispositivo que se está registrando a futuro.

```ruby
def create
  @user = User.new(name: "", token: SecureRandom.urlsafe_base64)

  if @user.save
    render json: @user.token, status: :created, location: @user
  else
    render json: @user.errors, status: :unprocessable_entity
  end
end
```

Figura 9 Registro de usuarios y obtención de token

En el aplication_controller del proyecto (Figura 10) se programa un callback que se lanzará para todas las acciones de todos los controladores y verificará el token.

```ruby
class ApplicationController < ActionController::API

  before_action :validate_token

  def validate_token
    Rails.logger.debug request.remote_ip
    unless User.where(token: params[:token]).any?
      head(403)
    end
  end

end
```

Figura 10 Validación de token mediante callback en AplicationController

De este modo ya tendremos montado el sistema de autenticación en la aplicación. Tan solo nos quedará evitar que se valide el token en el controlador de usuarios, ya que en el será en el que se registrará un nuevo usuario para conseguir el token. Pare ello solo utilizaremos la siguiente línea de código en el users_controller:

```
skip_before_action :validate_token
```

4.1.7.3 Figaro

Figaro es una gema para Ruby On Rails que ayuda securizar las aplicaciones del lado del servidor, evitando la inserción de claves y contraseñas en el código, lo cual facilita la tarea de hacer código libre y poderlo compartir fácilmente en repositorios públicos sin comprometer la seguridad de la aplicación en entornos de producción.

Figaro básicamente lo que hace es centralizar todas las claves y contraseñas de la aplicación en un único fichero que la gema al instalarse crea y añade automáticamente al gitignore para evitar compartir este fichero en nuestro repositorio de código.

La gema proporciona la infraestructura necesaria para centralizar las contraseñas en el archivo application.yml en el cual podemos asignar un nombre de variable a cada contraseña pudiendo usar estas variables en el código en lugar de las contraseñas, así por ejemplo en el archivo de configuración de la base de datos nos referiremos a la contraseña de esta como "ENV['db_pass']" y en el archivo application.yml (Figura 11) tendremos la siguiente asignación: "db_pass: 'miContraseña123'"

De este modo también estamos haciendo una mejor gestión de las contraseñas centralizándolas en un único fichero lo cual nos va a permitir cambiarlas fácilmente en un único lugar sin tener que estar haciendo una búsqueda en todo el código.

Esta gema nos permite definir distintas contraseñas según el entorno en el que nos encontremos, así podremos centralizar las contraseñas de los entornos de producción, desarrollo y test en un único fichero.

Figura 11 Ejemplo de un archivo application.yml de Figaro

4.1.7.4 Capistrano

Otra gema que usaremos en el proyecto es Capistrano, una gema que nos ayudará en la tarea de los despliegues a producción de nuestra aplicación.

Capistrano nos permite configurar varios stages, en los que podemos definir distintas configuraciones de despliegue, lo cual se utiliza típicamente para despliegues multientorno o multiservidor, aunque en este proyecto únicamente configuraremos un stage que llamaremos "production" para realizar los despliegues a la Raspberry.

Para realizar los despliegues de la aplicación además del stage deberemos configurar un pequeño script en el archivo deploy.rb, además de ciertos parámetros de configuración de Capistrano en el archivo Capfile.

La gema Capistrano una vez terminada la configuración nos permitirá hacer despliegues con un comando tan simple como `cap <nombre del stage> deploy` en nuestro caso como hemos llamado "production" a nuestro stage, nos bastará con `cap production deploy` para realizar un despliegue completo de la aplicación.

El funcionamiento de Capistrano sigue el esquema de la Figura 12

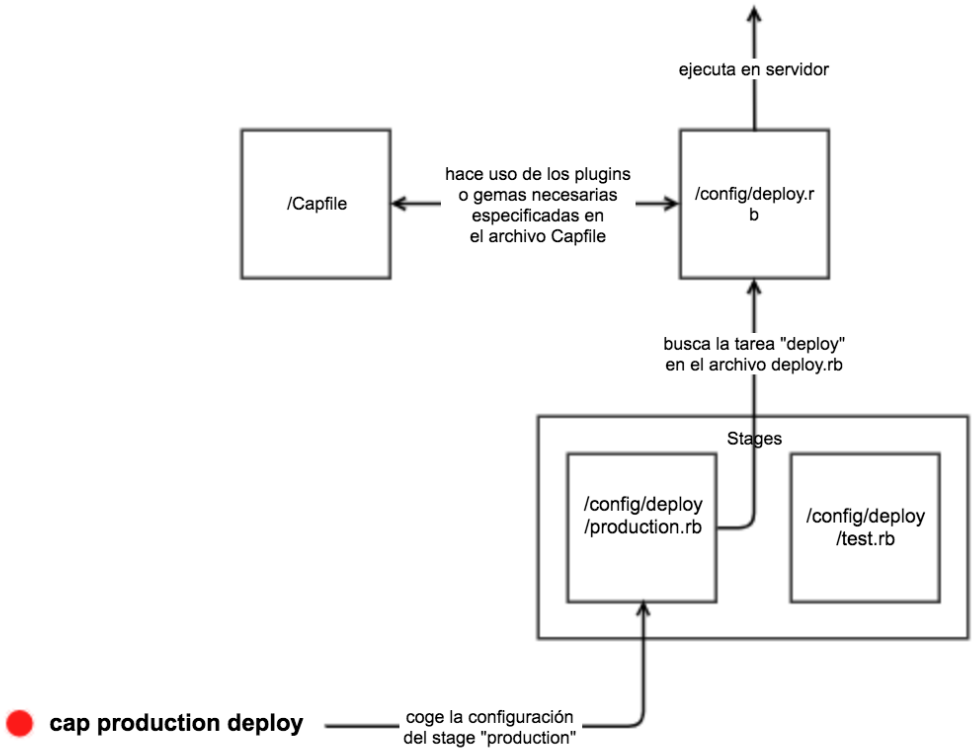


Figura 12 Diagrama de trabajo de Capistrano

Configuraremos Capistrano para que siga los siguientes pasos:

- Subir a la Raspberry el código desde la rama indicada de nuestro de repositorio en Github en un nuevo directorio con la fecha y hora del despliegue dentro del directorio releases.
- Instalar las gemas necesarias y sus dependencias con bundler.
- Realizar las migraciones de base de datos que estén pendientes
- Crear los directorios necesarios para la aplicación.
- Enlazar a la nueva reléase los archivos de configuración comunes contenidos en el directorio shared
- Reiniciar el servidor de aplicación Puma
- Mantener únicamente las 5 últimas releases borrando las sobrantes

Esta configuración la conseguiremos básicamente con tres archivos de Capistrano que se describirán a continuación, el archivo de stage, el Capfile.rb y el archivo de deploy.

Una vez configurado y desarrollado el script de Capistrano debemos configurar varios aspectos en el servidor.

El servidor que será nuestra Raspberry debe tener habilitado el ssh, también debemos tener instalado Ruby, Rails, Apache, Puma y todo lo necesario para que la aplicación funcione. Además lo más recomendable es crear un usuario específico para los despliegues, y que sea

ese usuario el que use Capistrano. Deberemos elegir el directorio donde se alojará la aplicación y conceder permisos sobre ese directorio al usuario de Capistrano.

4.1.7.4.1 Capistrano Stage

Un archivo de stage (Figura 13) contiene la configuración para lanzar tareas de Capistrano contra un entorno en específico. Contiene la dirección IP del servidor sobre el que lanzaremos las tareas, la configuración de acceso y conexión al mismo, o la rama del repositorio de código que se utilizará para ejecutar dichas tareas.

Capistrano permite lanzar diversos tipos de tareas contra entornos locales o remotos, siendo el eje principal de estas tareas el despliegue de aplicaciones.

Para realizar el despliegue del proyecto a la Raspberry que será el servidor que utilizaremos, necesitamos conocer su dirección IP, y definir el método de conexión con el mismo junto con la configuración del mismo. En este proyecto se usa una conexión ssh, con claves RSA.

```
role :app, %w{192.168.1.112}
role :web, %w{192.168.1.112}
role :db,  %w{192.168.1.112}
server '192.168.1.112', user: 'pi', roles: %w{web app}
set :ssh_options, {
    keys: %w(~/.ssh/id_rsa),
    forward_agent: false,
}

set :branch, :master
```

Figura 13 Archivo stage de Capistrano, config/deploy/production.rb

Es necesario también definir unos roles de acceso al servidor para mantener la seguridad e integridad del mismo, y por último se define también la rama del repositorio que utilizaremos en los despliegues, en este caso se desplegará siempre master.

4.1.7.4.2 Capistrano Capfile

```
# Load DSL and set up stages
require "capistrano/setup"

# Include default deployment tasks
require "capistrano/deploy"

#For the linked files
require 'capistrano/linked_files'

require "capistrano/scm/git"
install_plugin Capistrano::SCM::Git

require "capistrano/rvm"
require "capistrano/puma"
require "capistrano/puma/jungle"

require "capistrano/rails/migrations"

# Load custom tasks from `lib/capistrano/tasks` if you have any defined
Dir.glob("lib/capistrano/tasks/*.rake").each { |r| import r }
```

Figura 14 Archivo Capfile de Capistrano, /Capfile

El archivo Capfile (Figura 14) de Capistrano es el que concentra la configuración mediante plugins de la herramienta, siendo este el archivo al que deberemos añadir cualquier dependencia que necesitemos para nuestro despliegue.

Contamos con numerosos plugins para Capistrano que nos facilitaran las tareas de despliegue.

Por ejemplo algunos de los plugins que usaremos serán el de Capistrano git para facilitar el uso del repositorio de código o Capistrano puma que nos facilitará tareas de para el uso del servidor Puma.

4.1.7.4.3 Capistrano Deploy

El archivo deploy (Figura 15) de Capistrano es el que concentra todas las tareas que Capistrano será capaz de realizar, es aquí por lo tanto donde programaremos esas tareas.

Este archivo por lo general contiene configuración similar a la del stage pero más general, ya que será la configuración que utilicen todos los stages

Aquí será donde definamos las tareas que queremos usar. La principal tarea es la deploy, por lo tanto este archivo deberá tener una tarea con ese nombre, pero también podremos programar otras tareas en este archivo.

También es importante saber que podemos hacer uso de otras tareas que nos proporciona el propio Capistrano o gemas o plugins complementarios a Capistrano, así por ejemplo disponemos tareas de reinicio del servidor Puma gracias a la gema capistrano-puma.

```
lock "3.7.2"

set :application, "rasp_api"
set :repo_url, "git@github.com:javierrodriguez94/rasp_api.git"
set :resque_environment_task, true
set :pty, true

set :default_shell   , "/bin/bash -l"
set :rvm_ruby_version, "2.4.0"
set :rvm_type, :user

# Default deploy_to directory is /var/www/my_app_name
set :deploy_to, "/var/www/rasp_api" #'/home/USER/YOUR-APP-FOLDER'

# Default value for :linked_files is []
set :linked_files, fetch(:linked_files, []).push('config/application.yml', 'config/secrets.yml')

# Default value for keep_releases is 5
set :keep_releases, 5


namespace :deploy do
  desc 'Restart application'
  task :restart do
    on roles(:app), in: :sequence, wait: 5 do
      execute "/etc/init.d/puma restart" # assumes puma jungle tools installed
    end
  end
  task :install_dependencies do
    on roles(:web), in: :sequence, wait: 5 do
#execute("cd #{release_path} && rvm use 2.4.0 && bundle install")
      within release_path do
        execute :bundle , "--without development test"
      end
    end
  end
  after :published, :install_dependencies
end
```

Figura 15 Archivo deploy de Capistrano, /config/deploy.rb

4.1.7.5 Routes

En Rails disponemos de un sistema de rutas muy flexible y configurable que se implemente en un archivo de forma muy visual y sencilla.

El archivo que contiene las rutas de una aplicación Rails se encuentra en el directorio /config, y su nombre es routes.rb

En él se definen las rutas de la aplicación con una sintaxis clara y visible, en la que podemos configurar las rutas con bastante detalle, indicando el método, get o post que se usará, el controlador al que dirigirán y la acción del mismo que se debe ejecutar, alias para las rutas, o incluso restricciones sobre las mismas, pudiendo caparlas por ejemplo por un rango de IPs.

En la **Error! Reference source not found.** podemos observar que aspecto tiene el archivo de definición de rutas de este proyecto.

```
Rails.application.routes.draw do

  controller :lights do
    get "/lights/on" => :on
    get "/lights/off" => :off
    get "/lights/state" => :state
  end

  controller :camera do
    get "/camera" => :photo
  end

  controller :tasks do
    get "/tasks/cpu" => :cpu
    get "/tasks/disk" => :disk
    get "/tasks/ram" => :ram
    get "tasks/temp" => :temp
  end

  controller :sensors do
    get "sensors/temp" => :temp
    get "sensors/humidity" => :humidity
```

Figura 16 Archivo routes

4.1.7.6 Gema GPIO

En Ruby disponemos de una gran cantidad de gemas gracias a la gran comunidad de desarrolladores que posee, y eso nos ayuda a tener un código más limpio y encapsulado y a evitar desarrollar código de más bajo nivel.

En este caso vamos a hacer uso de una gema que nos facilitará el uso de la interfaz GPIO de la Raspberry. La gema se llama rpi-gpio y nos ofrece funciones para leer o escribir pines del GPIO, en este caso lo que vamos a necesitar es cambiar el valor de un pin que será el que controle los relés para encender o apagar las luces.

En la Figura 17 podemos ver el controlador que se encarga de gestionar el encendido y apagado de la luces, es algo bastante sencillo con dos métodos on y off que obtienen el pin en el cual está conectado el control del relé para las luces del archivo de Figaro, y llaman con ese valor a la función GPIO en la cual se implemente la lógica, en la que se comprueban los parámetros, se inicializa el pin y se le da valor low o high según deseemos encender o apagar las luces.

El caso concreto de la apertura de la puerta funciona de la misma manera, mediante un relé que controlado por su correspondiente controller en Rails y haciendo uso de la gema rpi-gpio será accionado para abrir la puerta con cierre automático.

```ruby
class LightsController < ApplicationController

  def on
    gpio ENV["LIGHT_PIN"], 0
    render json: "ok"
  end

  def off
    gpio ENV["LIGHT_PIN"], 1
    render json: "ok"
  end

  private

  def gpio pin, value
    pin = pin.to_i
    raise("GPIO Error - Invalid pin #{pin}\n") if pin > 40 or pin < 0
    RPi::GPIO.set_numbering :board
    RPi::GPIO.setup pin, :as => :output
    if value.to_s == "0"
      RPi::GPIO.set_low pin
    elsif value.to_s == "1"
      RPi::GPIO.set_high pin
    elsif value == nil
      RPi::GPIO.setup pin, :as => :input
      return Rpi::GPIO.input pin #Rpi::GPIO.low? pin ? 0 : 1
    else
      raise("Error invalid value")
    end
    return nil
  end

end
```

Figura 17 LightsController, uso de la gema rpi-gpio

4.1.7.7 Gema DHT-11

También disponemos de una gema específica para el sensor de temperatura y humedad que estamos utilizando, se trata de la gema dth-sensor-ffi, la cual nos permite leer fácilmente este sensor en Ruby.

Como en el caso del módulo de los relés y la gema de control del GPIO, en este caso lo correcto es tener un controlador para esta funcionalidad que se ha llamado SensorsController (Figura 18), ya que será el encargado de tomar medidas de los sensores.

```ruby
class SensorsController < ApplicationController

  require "dht-sensor-ffi"

  def temp
    render json: DhtSensor.read(ENV["DHT_PIN"].to_i, ENV["DTH_TYPE"].to_i).temp
  end

  def humidity
    render json: DhtSensor.read(ENV["DHT_PIN"].to_i, ENV["DTH_TYPE"].to_i).humidity
  end
end
```

Figura 18 SensorsController, uso de la gema DHT-11

El uso de esta gema como cualquier otra pasa por supuesto por su inclusión en el Gemfile del proyecto. Una vez incluida podremos usarla después de hacer un require de la misma en el controlador para indicar que será necesaria y se hará referencia a ella. Tras esto, dispondremos de una función read a la que se debe pasar como parámetros el pin al que está conectado el sensor y el tipo de sensor que estamos leyendo, ya que la gema es válida para dos tipos (el DHT-11 y el DHT-22), ambos datos los obtenemos con Figaro del aplication.yml y leemos los valores sacando de esta lectura fácilmente la temperatura y la humedad.

4.2 Dispositivos IoT

4.2.1 Relés

Un relé es un dispositivo electromagnético el cual podemos usar a modo de interruptor, ya que nos permite abrir o cerrar un circuito de una potencia grande usando señales de corriente muy pequeña.

En este caso un relé es útil ya que la Raspberry trabaja con 5 voltios y no podríamos abrir y cerrar circuitos domésticos de 220 voltios sin la ayuda de un relé.

De este modo con la Raspberry podemos controlar un relé a través de uno de sus pines del GPIO, sacando por el 5 voltios o 0 en función de si queremos activar el relé o no, pudiendo así controlar un circuito doméstico.

En este proyecto utilizaremos el módulo de la Ilustración 1 que proporciona cuatro relés que pueden ser controlados de forma independiente, y que será uno de los dispositivos IoT cableados.

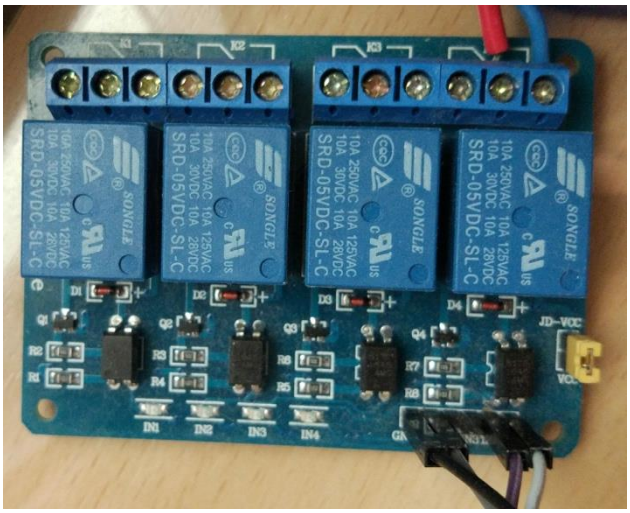


Ilustración 1 Modulo de relés utilizado en el proyecto

Con este módulo disponemos de cuatro relés, pero únicamente utilizaremos dos de ellos, los cuales conectaremos a una lámpara para poder controlar el encendido y apagado de las luces desde nuestra aplicación, y al sistema de apertura de la puerta. Para controlar dicho modulo debemos conectar un cable desde uno de los pines de 5 voltios de la Raspberry a la entrada VCC del módulo, otro cable de uno de los pines de tierra de la Raspberry a la entrada GND del módulo, y además uno de los pines de control numerados de IN1 a IN4 deberá ir conectado a un pin libre de la Raspberry, que será por el cual sacaremos 5voltios o 0 voltios para controlar el relé.

4.2.2 DHT-11

El DHT-11 es un sensor de temperatura y humedad digital y de bajo coste, el cual nos permite trabajar a 5 o a 3,3 voltios, lo cual nos evitaría un gran inconveniente si deseáramos usar este sensor como dispositivo IoT inalámbrico junto con el ESP-8266, ya que este último trabaja a 3,3v únicamente.

En el proyecto utilizaremos el DHT-11 como dispositivo IoT cableado por lo cual trabajaremos con los 5v que da la Raspberry.

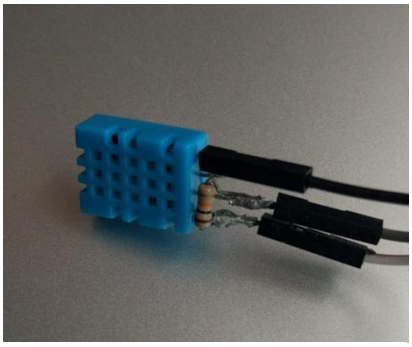


Ilustración 2 DHT-11 Sensor de temperatura y humedad digital

Otro inconveniente de este tipo de sensores a la hora de trabajar con Raspberry es que la mayoría suelen ser analógicos, lo que implica que representan la información en un rango de 0 a 1023 que debe ser leído mediante pines analógicos, pero la Raspberry carece de este tipo de pines. Por lo tanto el hecho de que este sensor sea digital en lugar de analógico evita este problema.

La conexión del DHT-11 con la Raspberry se realiza conectando la alimentación, 5 voltios, y un pin de tierra, de acuerdo al esquema de conexiones que encontramos en el datasheet del sensor [9]. Necesitaremos también conectar una resistencia de 10 omihos entre los pines VCC y la salida de datos. El pin contiguo al VCC es el pin de la salida de datos que conectaremos a la Raspberry y a través del cual podremos leer la temperatura y la humedad

4.2.3 ESP-8266

El ESP-8266 es un pequeño microcontrolador de bajo coste y tamaño y consumo reducidos que gracias a que cuenta con conectividad wifi será perfecto para crear dispositivos IoT inalámbricos.

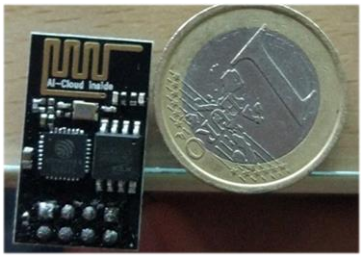


Ilustración 3 ESP-8266, comparación de su tamaño

A este microcontrolador podremos conectar un sensor de temperatura y/o humedad como el DHT-11, un sensor de movimiento, una cámara, o como en este caso un pulsador. Tiene gran potencial con diversos dispositivos convirtiéndolos en inalámbricos, así pues podemos tener por ejemplo, varios sensores de

temperatura conectados cada uno a un EPS-8266 distribuidos en distintos lugares de la casa, sin necesidad de cables, y registrando la temperatura de cada lugar cada cierto tiempo.

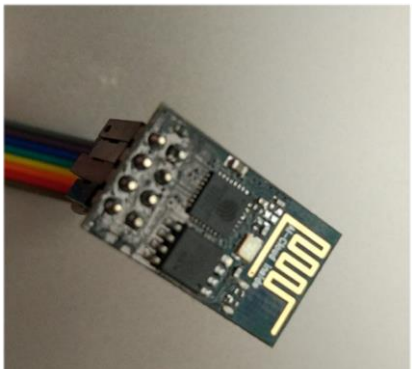


Ilustración 4 ESP-8286 modelo 01

El ESP-8266 funciona con 3,3 voltios por lo tanto usaremos una pequeña batería de este voltaje para alimentarlo.

En este proyecto utilizaremos el microcontrolador para dotar de conectividad inalámbrica a un pulsador, de modo que cuando este sea accionado el ESP-8266 envíe una petición HTTP a la Raspberry. El pulsador será el botón de un timbre o portero automático doméstico, de modo que cuando alguien llame a este, el microcontrolador enviará una petición a la Raspberry que notificará al teléfono del usuario.

Para ello haremos uso de un pulsador sencillo con una configuración de pull-down mediante una resistencia, de modo que cuando el pulsador sea accionado el ESP-8266 reciba una señal high (5v) en uno de sus pines GPIO que estará siendo monitorizado por el programa cargado en el microcontrolador, y cuando esa señal high sea recibida el microcontrolador mandará una petición HTTP a la Raspberry

Esto lo conseguiremos cargando en el ESP-8266 el programa que aparece en la Figura 19. Este programa se divide en dos partes una primera función llamada setup que será la primera que ejecute el microcontrolador, en la que se inicializa la comunicación serie, el pin 2 para lectura y la conexión wifi con el SSID y la contraseña. Tras esto se efectúa la conexión a la red wifi y se imprime por el monitor serie la IP. La segunda parte es una función loop, que es el bucle principal del programa que se estará ejecutando de forma infinita, en este bucle se lee el valor del pin 2 del GPIO, y si este está a 5 voltios, un valor high, se envía la petición HTTP.

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

#define URL "http://192.168.1.112/push"

HTTPClient http;

void setup()
{
  Serial.begin(115200);
  Serial.println();

  pinMode(2,INPUT);


  WiFi.begin("Wifi_Casa", "<PASSWORD>");

  Serial.print("Connecting");
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println();

  Serial.print("Connected, IP address: ");
  Serial.println(WiFi.localIP());
}


void loop() {
  if(analogRead(2)==1023){
    http.begin(URL); //HTTP
    http.GET();
  };
}
```

Figura 19 Programa cargado en el ESP-8266

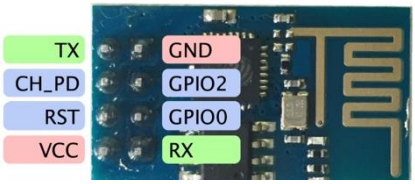


Figura 20 Esquema de conexiones ESP-8266

Es importante mencionar los tipos de arranque de los que dispone el ESP-8266, ya que podemos iniciarlo para correr el programa cargado en el o en modo de carga de programas en el que podremos cargar un nuevo script para que sea ejecutado por el microcontrolador. Toda la información necesaria sobre el ESP-8266 la podemos encontrar en su datasheet[14].

Para poner el microcontrolador en modo de carga deberemos tener el pin GPIO0 a high y el pin CH_PD debe estar también a high ya que es el que habilita el funcionamiento del microcontrolador, para arrancar en modo normal y ejecutar el programa cargado, este último pin estará a high también pero el GPIO0 estará a low. El esquema de conexiones se presenta en la Figura 20

Para cargar programas podemos usar un módulo cargador por USB directamente o como se ha hecho en este caso usar un Arduino Uno usando su conexión serial mediante los puerto Rx y Tx de ambos dispositivos. De este modo podremos cargar programas a través del IDE de Arduino conectando este mediante su cable USB al ordenador.

4.3 Aplicación móvil

Para la codificación de la aplicación móvil Android el lenguaje principal será Java y se usará Android Studio como IDE para facilitar el desarrollo.

En Android Studio podemos crear un proyecto para Android que nos generará una estructura básica de directorios y ficheros esenciales para una aplicación, y a partir de ahí comienza el desarrollo de esta aplicación.

Capturas de la aplicación en Anexo D.

4.3.1 Volley

Volley es la librería para peticiones HTTP para Android de Google, que encapsula el uso de las funciones de red y concretamente del protocolo HTTP de Android en una biblioteca que brinda funciones de alto nivel para simplificar el uso de estos protocolos.

Añadir la librería Volley al proyecto es relativamente sencillo, primero tendremos que descargar el archivo .jar compilado de Volley, o compilarlo descargando el código de los repositorios oficiales [11]. Una vez que tengamos el .jar de Volley se debe meter en el directorio libs de nuestro proyecto y a continuación añadir la siguiente línea en las dependencias del Gradle de aplicación del proyecto:

```
compile files('libs/volley.jar')
```

También debemos tener permisos de red para poder hacer uso de la misma en el terminal Android y poder así enviar peticiones HTTP. Para ello necesitaremos añadir estas líneas a nuestro Manifiest:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Para enviar una petición HTTP debemos declarar una cola de peticiones e inicializarla, una vez hecho esto tan solo hace falta crear una petición con el método StingRequest, donde podremos especificar si deseamos una petición vía get o post, la URL de destino de la petición, los datos de la petición y unos listeners para ejecutarse en caso de que la petición

se haga correctamente o surja algún error. Con la petición formada bastará añadirla a la cola inicializada anteriormente.

```
RequestQueue queue = Volley.newRequestQueue(activity);
StringRequest request = new StringRequest(Request.Method.GET, url, listener, errorListener);
queue.add(request);
```

Si la petición se realiza correctamente se llamará al primer listener en el cual se programarán las acciones necesarias en caso de éxito de la petición, y en caso de error se llamará al segundo donde podremos manejar el error.

Esta biblioteca nos facilita el uso del protocolo HTTP pudiendo así realizar fácilmente peticiones a la API que implementa la Raspberry para requerir servicios o información.

El funcionamiento de Volley en peticiones HTTP que devuelven respuestas en json como es el caso de la API de este proyecto, es relativamente sencillo, pero el uso de la librería se complica más cuando tratamos de recibir una imagen como respuesta a una petición HTTP. Este caso se da en este proyecto cuando intentamos obtener una imagen desde la webcam, la cual nos es devuelta como respuesta a cierta petición HTTP. La Raspberry toma una imagen de la webcam, la guarda en un archivo jpg y este archivo lo envía como respuesta a la petición, y mediante la librería Volley con la que hemos realizado la petición en Android deberemos manejar esta situación bastante más compleja que las respuestas en json.

Aunque Volley recibe correctamente la respuesta no era posible interpretar esta como una imagen de primeras y mostrarla en un imageView simplemente. Para conseguir esto se hace uso del componente ImageLoader y el NetworkImageView de Volley además de los siguientes fragmentos de código adaptados a Volley, que se han obtenido del artículo de TruitOn referenciado en [13].

```java
public class CustomVolleyRequestQueue {

    private static CustomVolleyRequestQueue mInstance;
    private static Context mCtx;
    private RequestQueue mRequestQueue;
    private ImageLoader mImageLoader;


    private CustomVolleyRequestQueue(Context context) {
        mCtx = context;
        mRequestQueue = getRequestQueue();

        mImageLoader = new ImageLoader(mRequestQueue,
                new ImageLoader.ImageCache() {
                    private final LruCache<String, Bitmap>
                            cache = new LruCache<~>(20);

                    @Override
                    public Bitmap getBitmap(String url) { return cache.get(url); }

                    @Override
                    public void putBitmap(String url, Bitmap bitmap) { cache.put(url, bitmap); }
                });
    }

    public static synchronized CustomVolleyRequestQueue getInstance(Context context) {
        if (null == null) {
            mInstance = new CustomVolleyRequestQueue(context);
        }
        return mInstance;
    }

    public RequestQueue getRequestQueue() {
        if (null == null) {
            Cache cache = new DiskBasedCache(mCtx.getCacheDir(), 10 * 1024 * 1024);
            Network network = new BasicNetwork(new HurlStack());
            mRequestQueue = new RequestQueue(cache, network);
            // Don't forget to start the volley request queue
            mRequestQueue.start();
        }
        return mRequestQueue;
    }

    public ImageLoader getImageLoader() { return mImageLoader; }

}
```

Figura 21 CustomVolleyRequestQueue class, solución para recibir imágenes como respuesta HTTP en Android. [13]

En la Figura 21 podemos observar la clase CustomVolleyRequestQueue donde se hace uso de ImageLoader para recibir los datos de la petición HTTP e interpretarlos como una imagen a partir de un BitMap.

Por otro lado en la vista deberemos hacer uso del elemento NetworkImageView que provee Volley, no podremos conseguir nuestro objetivo con un ImageView normal como se pretendía en un principio. Podemos ver su implementación en la Figura 22.

```xml
<com.android.volley.toolbox.NetworkImageView
    android:id="@+id/networkImageView"
    android:layout_width="match_parent"
    android:layout_height="300dp"
    android:layout_centerHorizontal="true"
    android:background="#000000"/>
```

Figura 22 NetworkImageView [13]

Por último solo nos quedará juntar todas las piezas realizando la llamada a través de las líneas de código que podemos encontrar en la Figura 23, en las que se crea una instancia del CustomVolleyRequestQueue, obtiene la URL de la llamada, y se realiza a través del

ImageLoader, para posteriormente insertar el resultado en el NetworkImageView con el método setImageUrl. Además podemos definir una imagen a mostrar en caso de error y una imagen a mostrar durante el tiempo de carga de la petición.

```
mImageLoader = CustomVolleyRequestQueue.getInstance(context)
        .getImageLoader();
//Image URL - This can point to any image file supported by Android
final String url = "http://" + getIp() + context.getResources().getString(R.string.route_image);
mImageLoader.get(url, ImageLoader.getImageListener(mNetworkImageView,
        R.mipmap.rasp_api, android.R.drawable
                .ic_dialog_alert));
mNetworkImageView.setImageUrl(url, mImageLoader);
return mImageLoader;
```

Figura 23 Petición http de imágenes [13]

4.3.2 Firebase

Firebase es la plataforma de desarrollo móvil de Google que ofrece distintas herramientas para desarrolladores controlables desde una consola de administración (Figura 24).

Uno de los servicios que ofrece Firebase es la posibilidad de enviar notificaciones push de forma controlada y sencilla para el usuario. Desde la consola de administración de Firebase podremos mandar directamente notificaciones a dispositivos , y ver el estado de las mismas, si han sido recibidas y leídas y distintas estadísticas más, relativas a las notificaciones.

Firebase también dispone de una API para poder hacer uso de sus herramientas mediante peticiones HTTP, que es lo que nos interesa en este proyecto, automatizar el envío de notificaciones push al dispositivo móvil para que la Raspberry sea capaz de realizar su envío como respuesta a un evento, y esto lo conseguiremos utilizando la API de Firebase.

La implantación de Firebase en este proyecto consta de dos partes, la implementación del servicio en la aplicación móvil, y por otra parte el desarrollo relativo al envío de notificaciones desde la Raspberry mediante la API de Firebase

Firebase dispone de distintos tipos de API para Android, iOs, Unity o C++, de modo que podemos integrar el servicio fácilmente en esos lenguajes, pero dado que no hay una API específica para el lenguaje que utilizaremos en nuestro servidor tendremos que usar la API web para enviar las notificaciones mediante peticiones HTTP.

Con el siguiente curl podremos mandar notificaciones push:

```
curl -X POST
--header "Authorization: key=<CLAVE DE SERVIDOR DE FIREBASE>"     -
-Header               "Content-Type:              application/json"
https://fcm.googleapis.com/fcm/send
-d "{\"notification\":{\"body\":\"MENSAJE\"},
     \"to\":\"<DESTINATARIO>"}"
```

Este curl será el que hagamos desde la Raspberry cuando recibamos una petición HTTP a la url adecuada para que la notificación llegue al dispositivo móvil y como respuesta a esta notificación el usuario podrá desde la aplicación móvil enviar otra petición HTTP para abrir

la puerta, que activará un relé siendo el funcionamiento idéntico al del encendido y apagado de las luces.

Para hacer uso de los servicios de Firebase lo primero será registrarnos en la plataforma y dar de alta nuestra aplicación. Después podremos acceder a la consola de administración donde tendremos acceso a las herramientas para desarrolladores que se proporcionan.

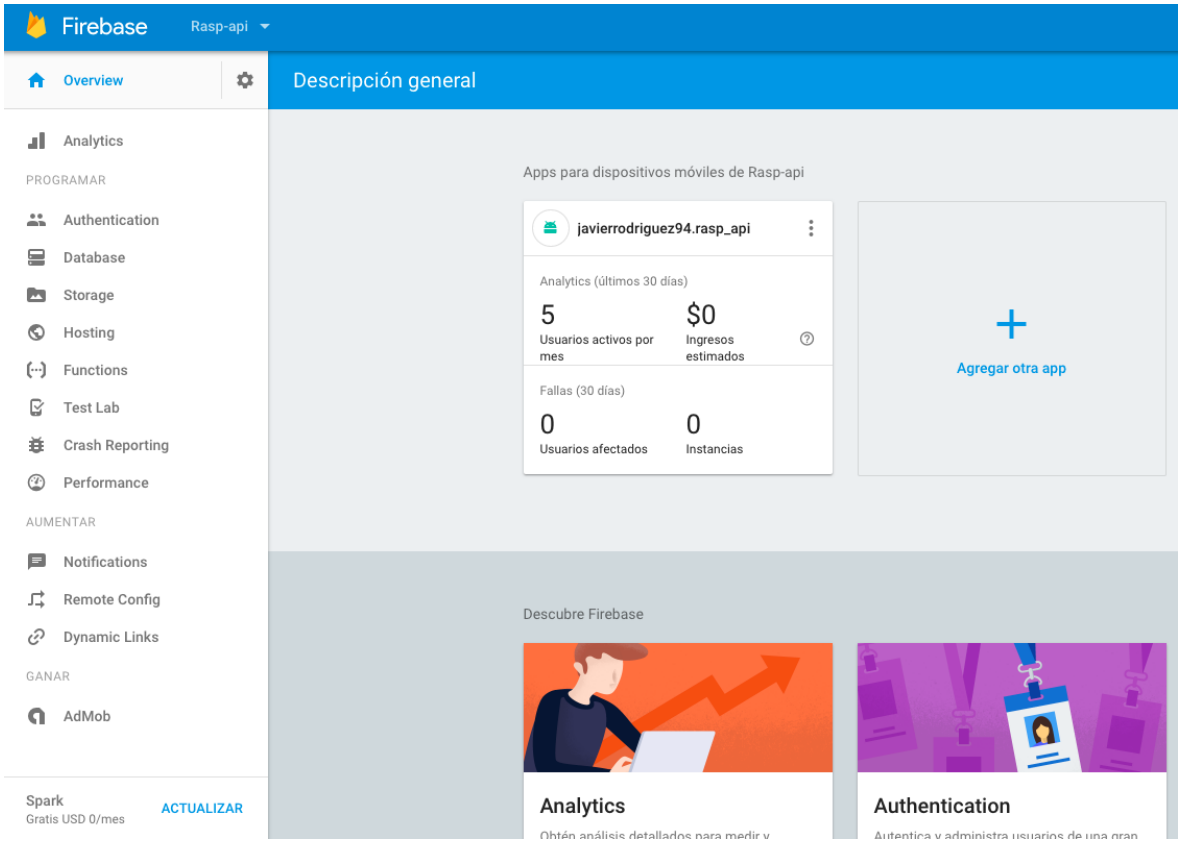


Figura 24 Consola de administración de Firebase

Desde la consola de administración podremos descargar un archivo de claves en formato json que deberemos meter en la raíz de nuestro proyecto Android.
En la web de Firebase nos proporcionan toda la documentación[15] sobre la plataforma y su integración con cualquier proyecto Android, simplemente hay que seguir unos pasos básicos para empezar a utilizar Firebase, que consisten en añadir Firebase y los servicios de Google en el Gradle de nuestra aplicación.

Siguiendo la documentación de Firebase[15], debemos añadir dos servicios a nuestra aplicación, tal y como vemos en la Figura 25. Una vez añadidos los servicios al manifiesto de nuestra aplicación debemos crearlos según con la funcionalidad que indica la documentación.

El primer servicio FirebaseIDService (Figura 26), actualiza el token de conexión con Firebase y se registra en su servidor para poder empezar a recibir notificaciones.

El segundo servicio (Figura 27), es el cual recibirá los mensajes enviados desde Firebase permitiéndonos manejarlos para mostrar notificaciones a partir de ellos. Dispone de un

método onMessageReceived, que recibe como argumento el mensaje de Firebase y gestiona su contenido, para mostrarlo en forma de notificación gracias al NotificationManager de Android

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

Figura 25 Modificación al manifiesto con Firebase [15]

```
public class FirebaseIDService extends FirebaseInstanceIdService {
    private static final String TAG = "FirebaseIDService";

    @Override
    public void onTokenRefresh() {
        // Get updated InstanceID token.
        String refreshedToken = FirebaseInstanceId.getInstance().getToken();
        Log.d(TAG, "Refreshed token: " + refreshedToken);

        // TODO: Implement this method to send any registration to your app's servers.
        sendRegistrationToServer(refreshedToken);
    }
```

Figura 26 FirebaseIDService [15]

```
public class MyFirebaseMessagingService extends FirebaseMessagingService {
    private static final String TAG = "FCM Service";
    @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {

        NotificationManager notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
        NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this);
        mBuilder.setSmallIcon(R.mipmap.ic_launcher);
        mBuilder.setContentTitle("t");
        mBuilder.setContentText("m");
        mBuilder.setStyle(new NotificationCompat.BigTextStyle().bigText("m2"));
        mBuilder.setTicker("ms");

        mBuilder.setDefaults(Notification.DEFAULT_LIGHTS);
        mBuilder.setDefaults(Notification.DEFAULT_SOUND);
        mBuilder.setDefaults(Notification.DEFAULT_VIBRATE);

        notificationManager.notify(1, mBuilder.build());

        Log.d(TAG, "From: " + remoteMessage.getFrom());
        Log.d(TAG, "Notification Message Body: " + remoteMessage.getNotification().getBody());
    }
}
```

Figura 27 FirebaseMessagingService [15]

5 Integración, pruebas y resultados

Las pruebas del sistema se dividen en varias partes, por un lado el software se ha probado con test unitarios sobre las funciones no triviales, por otro lado se han realizado pruebas de integración del sistema completo con usuarios reales con objetivo de testear la funcionalidad básica. En cuanto al hardware específicamente se ha probado su funcionamiento en conjunto con el software en los test de integración.

Los test de integración se llevan a cabo en un entorno de producción con un dispositivo Android virtual y el resto del sistema configurado con un entorno de producción totalmente operativo. En estas condiciones se conecta a través de una terminal y via ssh al servidor (la Raspberry) y se monitorizan en tiempo real los log de apache, de Rails con los siguientes comandos:

```
tail -f /var/log/apache2/error_log
tail -f <ruta aplicación>/current/logs/production.log
```

Tras esto podemos empezar a usar el sistema normalmente analizando los logs (Figura 28).

En cuanto a los test unitarios se ha usado Rspec para los test de la aplicación Rails. Rspec es la suite de test más usada en proyectos Ruby, y nos permite diseñar test y ejecutarlos manualmente o de forma automatizada además de facilidades para implementaciones con integración continua.

Los test de integración serán los que probaran la funcionalidad básica y verificarán el cumplimiento de los requisitos del apartado 3 de este documento, probando a su vez el correcto funcionamiento de toda la infraestructura montada para el correcto funcionamiento del sistema.

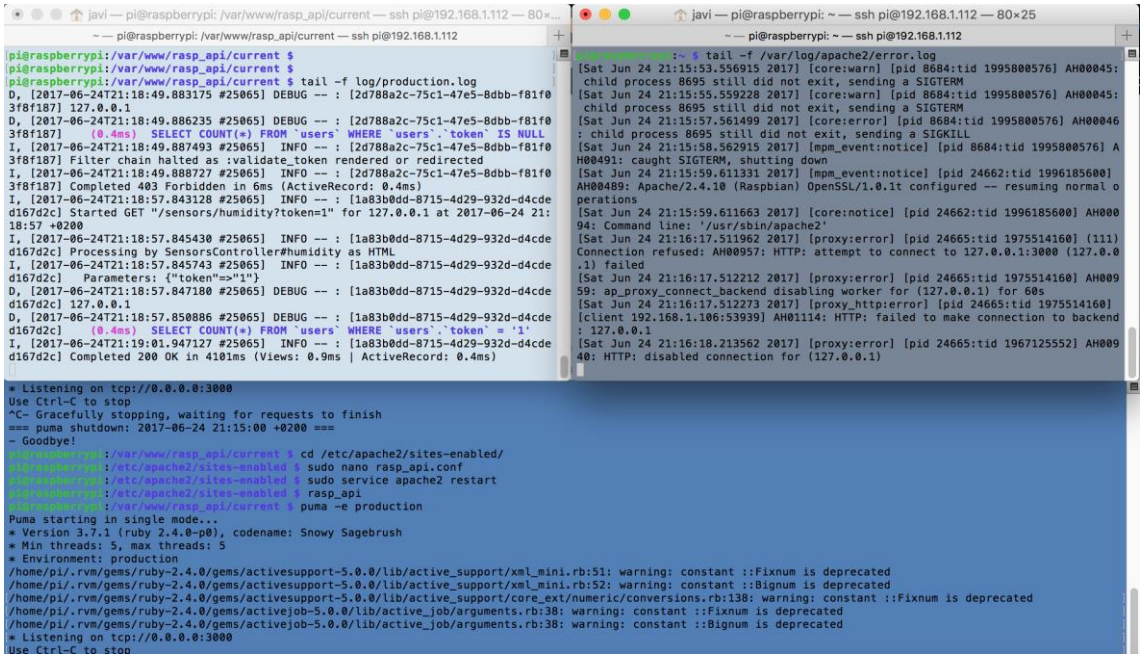


Figura 28 Seguimiento de logs en tests de integración

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Como conclusiones obtenidas fruto de este trabajo, tenemos la posibilidad de realización de un sistema domótico a partir de elementos de hardware libre y de bajo coste y uso de software libre, constituyendo este trabajo un proyecto de software y hardware libre que puede ser reproducido fácilmente.

Es interesante ver las observaciones sobre el internet de las cosas que hemos llevado a cabo a lo largo de esta memoria, como la facilidad de convertir cualquier dispositivo con una funcionalidad particular en un dispositivo conectado del internet de las cosas sin necesidad de una gran inversión.

La conectividad ha llegado a todo objeto susceptible de aportar información y es que es tan sencillo como conectarlo a un microcontrolador que le permita procesar datos y le otorgue conectividad a la red para tener un dispositivo IoT enviando y recibiendo datos.

En este proyecto se han trabajado dispositivos IoT cableados e inalámbricos con la intención de estudiar ambas implementaciones de las cuales podemos concluir que los dispositivos cableados aportan autonomía pero carecen de movilidad, mientras que los inalámbricos son fácilmente implementables gracias a microcontroladores como el ESP-8266, y suponen un incremento del coste económico y de complejidad respecto a los dispositivos cableados.

Se ha trabajado con entornos de producción y gestión de servidores configurando un servidor web junto con uno de aplicación, lo cual implica una gran tarea de sincronización de configuraciones y gestión de permisos de acceso para los distintos ficheros implicados, así como la configuración oportuna para automatizar cada despliegue sin riesgos y con un mecanismo de rollback.

Se ha montado toda una infraestructura hardware y software en perfecta armonía para albergar servidores, base de datos, tareas programadas, lógica, accesibilidad, seguridad, movilidad y acceso remoto.

6.2 Trabajo futuro

Como trabajo futuro sobre este proyecto sería interesante trabajar sobre la idea de dotar de una mayor complejidad a los dispositivos permitiendo el envió bidireccional de datos, por ejemplo una posible aplicación de esto sería poder enviar audio y video bidireccionalmente entre la aplicación móvil y un dispositivo IoT conectado al portero automático de la puerta.

Otra posibilidad a considerar seria añadir una interfaz de gestión web a la aplicación vía web, lo cual con el actual diseño del proyecto y las facilidades que aporta Rails en este aspecto seria relativamente sencillo gracias al planteamiento de escalabilidad que se ha tenido presente a lo largo de todo el proyecto.

Otras vías de trabajo podrían ir orientadas a la creación de tareas programadas desde la aplicación que puedan ser configuradas para ejecutarse periódicamente , en determinados momentos o como respuesta a un evento, y realicen tareas domóticas como pueden ser bajar las persianas a ciertas horas, encender las luces si se detecta movimiento o encender la calefacción si se baja de cierta temperatura.

Por último sería conveniente trabajar también la experiencia de usuario de la aplicación móvil creando interfaces más intuitivas y fáciles de utilizar.

Referencias

[1] Arik Gabay, Kevin Ashton Describes "the Internet of Things", Enero 2015, Smithsonian magazine, http://www.smithsonianmag.com/innovation/kevin-ashton-describes-the-internet-of-things-180953749/

[2] Ricardo Vega, El Internet de las Cosas como nuevo paradigma, Noviembre 2014, https://ricveal.com/blog/internet-de-las-cosas/

[3] Tesa Entr, http://www.tesa-entr.com/cerradura-inteligente/

[4] Iomando, https://www.iomando.com/

[5] Parkingdoor, https://parkingdoor.com/

[6] Phillips Hue, http://www.philips.es/c-m-li/hue

[7] Chema Alonso, La ciberseguridad en IoT vista por Chema Alonso, Abril 2016, IDGTV, http://www.idgtv.es/entrevistas/la-ciberseguridad-en-iot-vista-por-chema-alonso

[8] https://www.google.es/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwiZzZezuvXTAhVFRhQKHZzoAHEQjRwIBw&url=http%3A%2F%2Fwww.thingscity.com%2Fbosch-lanza-sus-servicios-en-la-nube-para-iot%2F&psig=AFQjCNG_N4WeFbA_By3FXB9vnTmDKscMoA&ust=1495059879083267

[9] Aosong(Guangzhou) Electronics Co.,Ltd, DHT11 Datasheet, https://akizukidenshi.com/download/ds/aosong/DHT11.pdf

[10] StatsCounter, Operating System Market Share Worldwide, May 2016 to May 2017, http://gs.statcounter.com/os-market-share

[11] Volley, Google, https://github.com/google/volley

[12] RVM, Ruby Version Manager, Copyright © 2009-2011 Wayne E. Seguin © 2011-2017 Michal Papis © 2016-2017 Piotr Kuczynski, https://rvm.io/

[13] Android Volley ImageLoader and NetworkImageView Example, Mohit Gupt, Truiton, http://www.truiton.com/2015/03/android-volley-imageloader-networkimageview-example/

[14] ESP8266 Datasheer, Espressif Systems, https://nurdspace.nl/images/e/e0/ESP8266_Specifications_English.pdf

[15] Documentación de Firebase, Google, https://firebase.google.com/docs/

Glosario

API	Application Programming Interface
Datasheet	Documento de especificaciones técnicas de un módulo o componente
Framework	Entorno de trabajo que proporciona una serie de herramientas
Gema	Nombre que se a las librerias en Ruby
GPIO	General Purpose Input/Output
High	Valor con el que se designa en electrónica al valor Vcc por lo general 5 voltios
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
Migración	Cambio en base de datos encapsulado en un pequeño script
Low	Valor con el que se designa en electrónica al valor GND, 0 voltios
Overclock	Aumentar la velocidad de reloj de un procesador para obtener mayor rendimiento
RVM	Ruby Version Manager
Rx	Abreviación usada en electrónica para referirse a la recepción de datos
Tx	Abreviación usada en electrónica para referirse a la transmisión de datos
URL	Uniform Resorce Locator

Anexos

A Manual de uso

El sistema domótico está provisto de una configuración inicial que no precisa de ser cambiada, de modo que se puede empezar a utilizar sin necesidad de cambios en la misma.

Emparejar los dispositivos:

El medio de interacción del usuario será un dispositivo Android con la aplicación instalada. El primer paso tras abrir la aplicación será emparejar nuestro dispositivo móvil con la centralita.
En la primera pantalla se nos dará la opción de introducir la dirección en la que está la centralita, la cual puede ser una dirección IP local, o una URL en caso de que tengamos una configuración con una URL apuntando a la IP de la centralita.

Tras introducir la IP/URL pulsaremos el botón de emparejar y el dispositivo guardará dicho dato para conexiones futuras.

Menú Principal:

Disponemos de un menú con 6 iconos para cada una de las tareas que ofrece el sistema.

Temperatura: obtener la temperatura en la casa a través del termómetro digital instalado en el sistema.

Humedad: podremos ver en pantalla la humedad relativa en la casa gracias al sensor de humedad.

Luz: con un interruptor en la pantalla de nuestro móvil podremos apagar las luces o dispositivos que tengamos configurados en el sistema.

Puerta: igual que en el caso anterior , en esta pantalla dispondremos de un interruptor para abrir la puerta.

Cámara: Obtendremos en pantalla una imagen desde la cámara integrada en el sistema.

Opciones: Un pequeño menú se abrirá mostrando las opciones de configuración.

Opciones:

En el apartado de opciones encontraremos parámetro configurables desde el dispositivo móvil, como por ejemplo la IP con la que está emparejado el dispositivo, que podremos cambiarla en caso de que la IP de la centralita cambie, o queremos usar una URL.

B Manual del programador

El software de este proyecto se encuentra repartido en la API en Ruby On Rails y en la aplicación móvil Android, este manual por tanto se dividirá en dichas partes

Manual del programador: API Ruby On Rails.

La API está programada según el patrón de diseño MVC que sigue el framework Rails, con distintos controladores diferenciados según la funcionalidad que maneja cada uno, así encontramos :

ApplicationController: Controlador principal de toda aplicación Rails encargado de la funcionalidad común de toda la aplicación, que contendrá la lógica aplicable a todos los demás controladores.

CameraController: Controlador encargado de las funciones de la cámara, en este caso hace uso de uns cámara web tomando una foto y renderizandola desde el directorio donde es guardada. Cualquier lógica relativa al procesamiento de la imagen o el uso de la cámara deberá ir en este controlador.

LightsController: Se encarga del control de las luces mediante el uso de la gema GPIO para el control de relés, cualquier evolutivo referente a la iluminación deberá parar por el uso de relés y su control en este controlador.

SensorsController: Gestiona los sensores del sistema obteniendo datos de los mismos mediante el uso de las gemas adecuadas para cada sensor. Los nuevos sensores implementados deberán gestionarse aquí.

TasksController: Procesa tareas internas del sistema, reportando datos del propio servidor como pueden ser su temperatura, carga de CPU o uso de disco. Es el encargado de centralizar la ejecución de comandos en el sistema.

UsersController: Lógica relativa a creación y gestión de usuarios en la aplicación

Al tratarse de una API carece de vistas como tal ya que los datos son renderizados en formato json directamente en el controlador debido a su simplicidad, en caso de ser requerido un mayor número de datos o mayor complejidad de los mismo lo correcto sería utilizar vistas en formato json.

En caso de implementar una interfaz visual vía web del mismo modo se implementarían vistas que se dividirían en vistas html para la parte de interfaz y vistas json para la parte de la API las cuales no darían ningún tipo de conflicto gracias a la gestión de formatos de renderización de vistas de Rails.

En cuanto los modelos únicamente se persiste en base de datos un token de usuario que se modeliza en el modelo Usuario gestionado mediante ActiveRecord.

Manual del programador: Aplicación Android.

La aplicación Android está programada en lenguaje Java en el IDE Android Studio, y siguiendo el patrón MVC. La aplicación carece de base datos ya que no es necesario registrar ni persistir ningún tipo de dato, únicamente es necesario guardar el token de conexión y se guardará como una preferencia.

La aplicación se estructura en varios Activities principales, que son PairActivity, MainActivity y PreferencesActivity, y otros Activities secundarios que corresponden a las distintas funciones que realiza el sistema, y son por ejemplo LightActivity, ImageActivity o TempActivity entre otros.

Los activitities principales tienen las siguientes funciones:

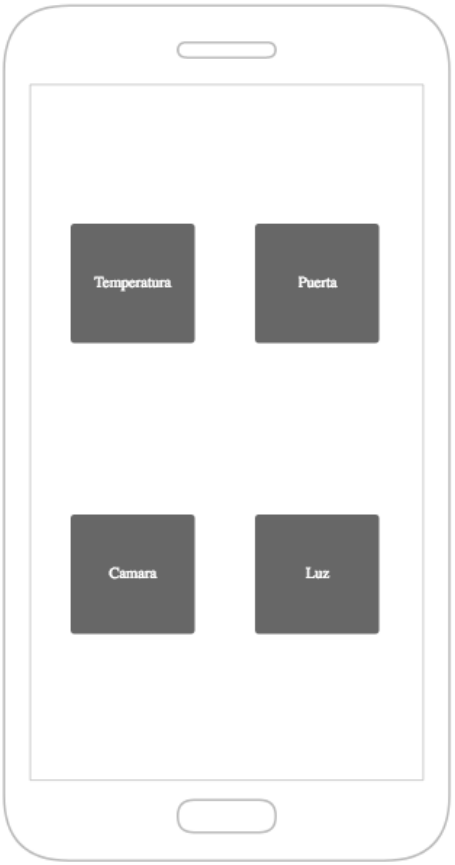
PairActivity: Solicita al usuario la IP o URL de la Raspberry y envía una petición a la misma para registrar a un nuevo dispositivo. Si la petición termina correctamente se recibe un token que la aplicación guardará como preferencia junto con la IP o URL

MainActivity: Es la actividad principalde la aplicación, que muestra el menú desde el que se puede acceder a todas las demás activities y opciones de la aplicación.

PreferencesActivity: Permite cambiar las preferencias de la aplicación.

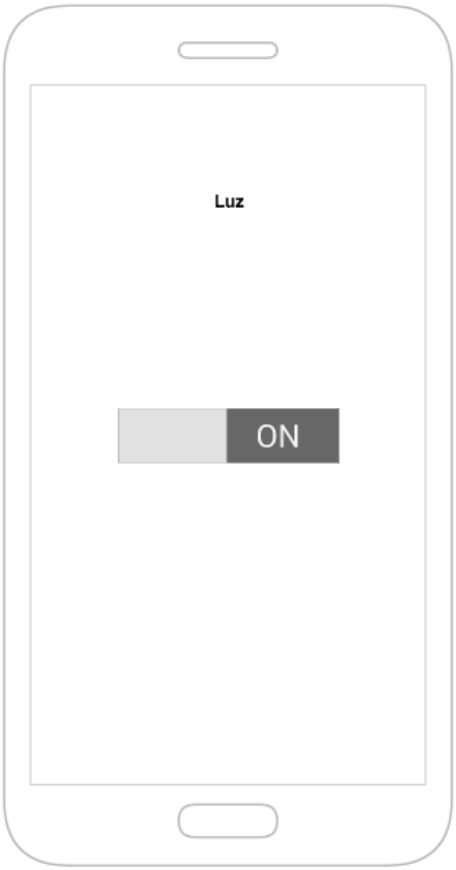
Por otro lado cabe destacar la clase Petición en la que se ha encapsulado la lógica de las peticiones HTTP mediante la librería Volley, de modo que cada vez que se hace una petición de este tipo en la aplicación se hace a través de esta clase.

C Maquetas de la aplicación móvil

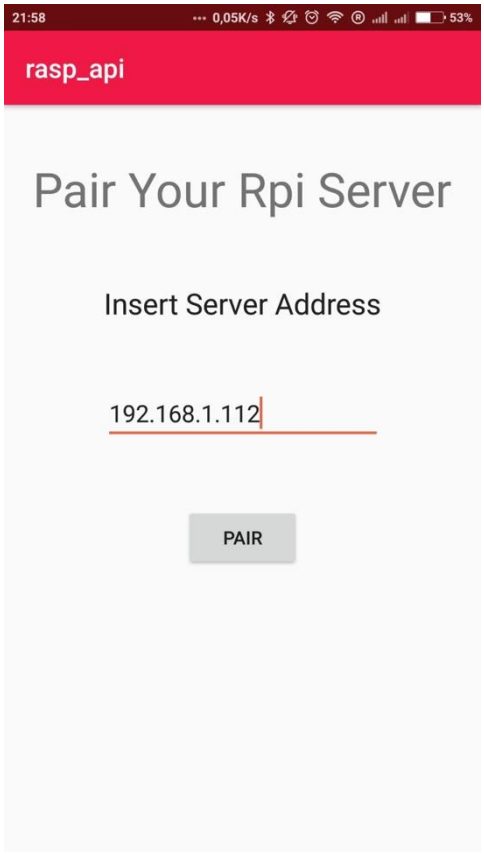


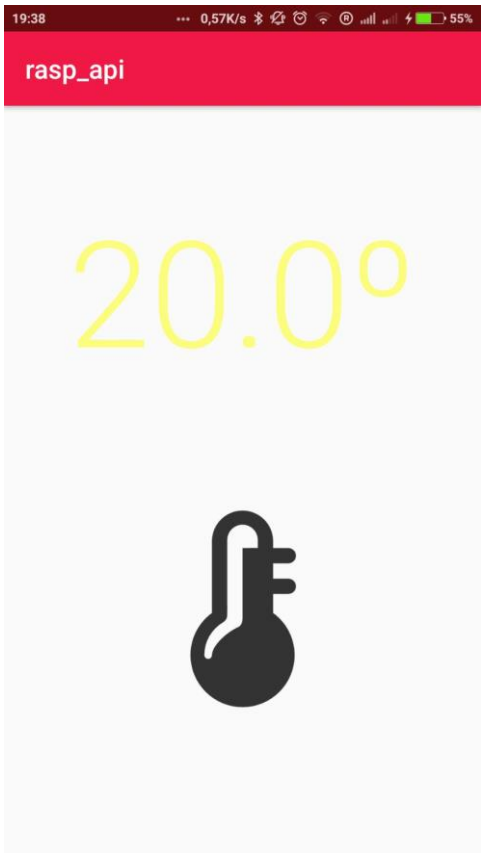

Camara

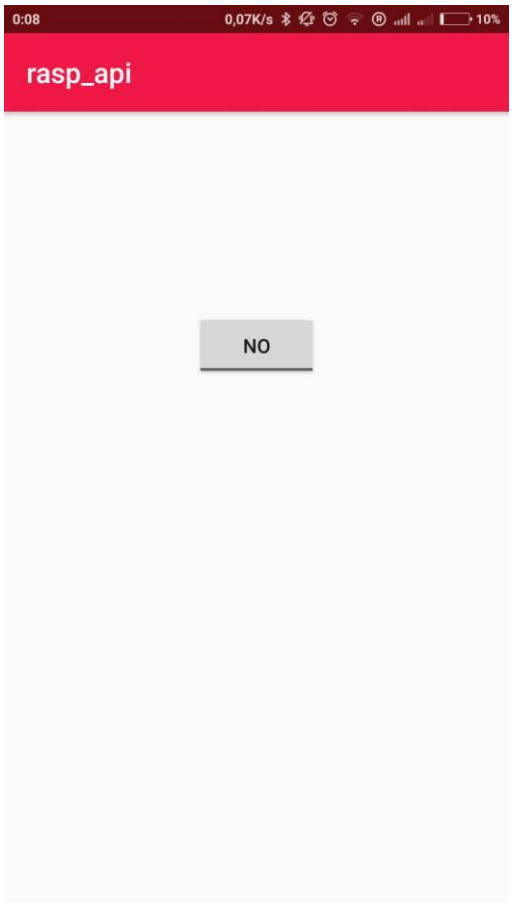

Temperatura
22º

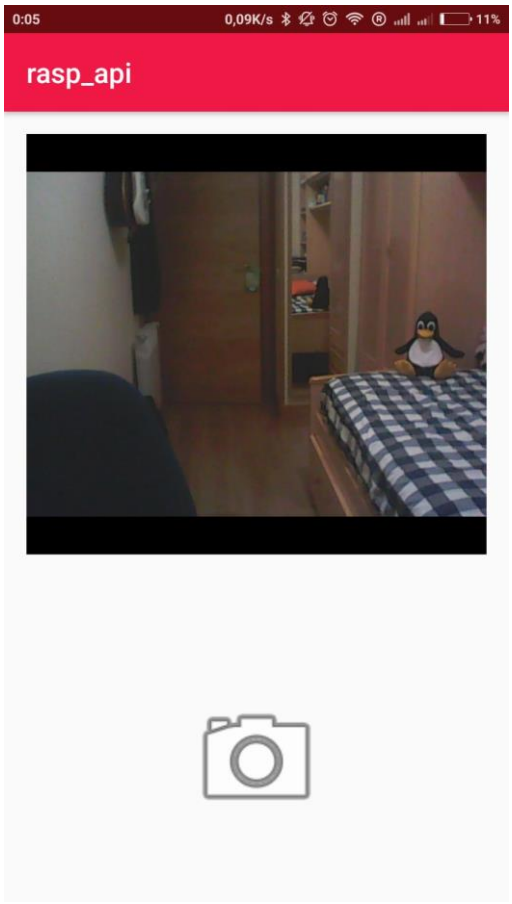

Luz
ON

D Capturas de la aplicación móvil






0:08
0,07K/s
10%
rasp_api
NO


0:05
0,09K/s
11%
rasp_api