



## **IMPLEMENTACIÓN DE UN MANEJADOR DE RECURSOS CON INTELIGENCIA AMBIENTAL PARA UNA VIVIENDA DEL SOLAR DECATHLON LATIN AMERICA 2019**

JUAN DAVID RAMÍREZ VILLEGAS

UNIVERSIDAD DEL VALLE  
FACULTAD DE INGENIERÍA  
ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA  
PROGRAMA ACADÉMICO DE INGENIERÍA ELECTRÓNICA  
25 de Octubre de 2019



## **IMPLEMENTACIÓN DE UN MANEJADOR DE RECURSOS CON INTELIGENCIA AMBIENTAL PARA UNA VIVIENDA DEL SOLAR DECATHLON LATIN AMERICA 2019**

JUAN DAVID RAMÍREZ VILLEGAS  
CÓDIGO: 1427249

ÉDINSON FRANCO MEJÍA, Dr.-Ing.  
FABIO GERMÁN GUERRERO, M.Sc.

UNIVERSIDAD DEL VALLE  
FACULTAD DE INGENIERÍA  
ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA  
PROGRAMA ACADÉMICO DE INGENIERÍA ELECTRÓNICA

25 de Octubre de 2019

# Índice

<b>Resumen</b>	<b>6</b>
<b>1 Marco Teórico</b>	<b>8</b>
1.1 Arquitectura . . . . .	8
1.2 Aplicación de escritorio. . . . .	8
1.3 Aplicación web. . . . .	9
1.4 Servidor . . . . .	9
1.5 Tcp y Tls . . . . .	9
1.6 Jwt . . . . .	10
1.7 Mqtt . . . . .	10
1.8 Http . . . . .	10
1.9 Serverless . . . . .	11
1.10 Backend . . . . .	11
1.11 Frontend . . . . .	12
1.12 Dispositivo . . . . .	12
1.13 Teoría de colores . . . . .	12
<b>2 Introducción</b>	<b>14</b>
2.1 Contexto introductorio . . . . .	14
2.2 Lineamientos de desarrollo . . . . .	14
<b>3 Diseño del sistema</b>	<b>17</b>
3.1 Diseño conceptual . . . . .	17
3.2 Diseño final . . . . .	17
<b>4 Api de animación</b>	<b>23</b>
4.1 Investigación y conceptualización . . . . .	23
4.2 Api de java . . . . .	24
4.3 Api de Python y Qt . . . . .	29
<b>5 House Manager</b>	<b>32</b>
5.1 Especificaciones técnicas . . . . .	32
5.2 Requerimientos Funcionales . . . . .	33
5.3 Funciones adicionales . . . . .	35
5.4 Algoritmos Utilizados . . . . .	37
5.4.1 Scheduler Handler . . . . .	38
5.4.2 Remote Mode Handler . . . . .	38
5.4.3 Mail Notification Handler . . . . .	38
5.4.4 Data Report Handler . . . . .	38
5.4.5 Indicator Handler . . . . .	38

<b>6 Serverless backend</b>	<b>40</b>
6.1 Especificaciones técnicas . . . . .	41
6.2 Base de datos relacional y no relacional . . . . .	42
6.3 Relación de requerimientos funcionales . . . . .	46
6.4 Funciones de nube . . . . .	47
6.4.1 Get_historical_data: . . . . .	47
6.4.2 Send_command: . . . . .	50
6.4.3 IOT_gateway_control: . . . . .	53
6.4.4 IOT_gateway_measure: . . . . .	54
<b>7 User app</b>	<b>56</b>
7.1 Especificaciones técnicas . . . . .	57
7.2 Requerimientos Funcionales . . . . .	59
7.3 Funciones adicionales . . . . .	61
<b>8 Pruebas y resultados</b>	<b>65</b>
8.1 Pruebas unitarias . . . . .	65
8.2 Pruebas de integración . . . . .	66
8.3 Resultados . . . . .	68
<b>9 Conclusiones y Trabajos futuros</b>	<b>70</b>
9.1 Conclusiones . . . . .	70
9.2 Trabajos futuros. . . . .	71
9.2.1 Pruebas con más dispositivos. . . . .	71
9.2.2 Integración de sensores y actuadores al sistema. . . . .	71
9.2.3 Interacción con la interfaz de voz de Google usando “Dialog flow”. . . . .	71
<b>Referencias</b>	<b>72</b>

## Índice de figuras

Figura 1	Primer diseño generalizado de la estructura del sistema Fuente: propia . . . . .	18
Figura 2	Diagrama generalizado de las entidades software del proyecto. Fuente: propia . . . . .	20
Figura 3	Diagrama de secuencia general de la comunicación. Fuente: propia . . . . .	22
Figura 4	Conceptualización inicial del api en sitio. Fuente: propia . . . . .	24
Figura 5	Estadísticas de los lenguajes de programación con más demanda. Fuente: [1] . . . . .	25
Figura 6	El diseño de estados del botón interactivo. Fuente: propia . . . . .	26
Figura 7	El diseño de los estados del botón interactivo desplegable. Fuente: propia . . . . .	26
Figura 8	Movimiento según la ubicación de la función horizontal. Fuente: propia . . . . .	26
Figura 9	Primer Aproximación del diseño de la interfaz en sitio. Fuente: propia . . . . .	27
Figura 10	Ajuste de diseño de la interfaz en sitio. Fuente: propia . . . . .	27
Figura 11	Versión final del concepto de diseño gráfico. Fuente: propia . . . . .	28
Figura 12	Botón de tamaño flexible genérico de la api de qt. Fuente: propia . . . . .	30
Figura 13	Boton de tamaño fijo usando imágenes y slider. Fuente: propia . . . . .	31
Figura 14	Visualización de la escalabilidad del sistema con una App no desarrollada por completo. Fuente: propia . . . . .	33
Figura 15	Escena del home del House Manager con barras de progreso para la indicación dinámica. Fuente: propia . . . . .	33
Figura 16	Escena de control del House Manager con sus calendarios. Fuente: propia . . . . .	35
Figura 17	Escena de medición del House Manager, con su respectivo panel para guardar los datos. Fuente: propia . . . . .	36
Figura 18	Escena de configuración del dispositivo House Manager. Fuente: propia . . . . .	36
Figura 19	Escena para el inicio de sesión del House Manager. Fuente: propia . . . . .	37
Figura 20	Descripción de los tiempos de muestreo utilizados en el sistema de adquisición. Fuente: propia . . . . .	39
Figura 21	Estadística de la repartición del mercado de servicios de nube en 2019. Fuente: [2] . . . . .	40
Figura 22	Diagrama general de la base de datos relacional MySql. Fuente: propia . . . . .	43
Figura 23	Grupo de tablas: users data. Fuente: propia . . . . .	44
Figura 24	Grupo de tablas: device info . . . . .	45
Figura 25	Grupo de tablas: device data. Fuente: propia . . . . .	46

---

Figura 26	Algoritmo de la función: Get historical data. Fuente: propia . . . . .	49
Figura 27	Algoritmo de la función: Send command. Fuente: propia . . . . .	52
Figura 28	Algoritmo de la función: Iot gateway control. Fuente: propia . .	54
Figura 29	Algoritmo de la función: Iot gateway measure. Fuente: propia .	55
Figura 30	El menú desplegable con las aplicaciones actualmente desplegadas. Fuente: propia . . . . .	58
Figura 31	Escena para la medición en tiempo real de las variables. Fuente: propia . . . . .	60
Figura 32	Escena de configuraciones, actualmente no disponible. Fuente: propia . . . . .	61
Figura 33	Escena para la visualización del consumo histórico. Fuente: propia . . . . .	62
Figura 34	Escena para el control de los circuitos de la vivienda. Fuente: propia . . . . .	63
Figura 35	Escena de bienvenida 1. Fuente: propia . . . . .	63
Figura 36	Escena de inicio de sesión. Fuente: propia . . . . .	64
Figura 37	Dispositivo Inelca utilizado para la prueba de concepto. Fuente: propia . . . . .	66
Figura 38	Prueba de concepto usando todos los sistemas. Fuente: propia .	67

## Índice de cuadros

Tabla 1	Requerimientos Funcionales del sistema . . . . .	16
---------	--	----

# **IMPLEMENTACIÓN DE UN MANEJADOR DE RECURSOS CON INTELIGENCIA AMBIENTAL PARA UNA VIVIENDA DEL SOLAR DECATHLON LATIN AMERICA 2019**

En este documento se explicará el proceso realizado para implementar un modelo cliente servidor, que comunique diferentes dispositivos bajo el paradigma de internet de las cosas. El sistema consta principalmente de 3 productos: el producto del backend, una aplicación móvil y una aplicación de escritorio. La aplicación móvil se encargó, a grandes rasgos, de monitorear y controlar de manera remota todos los dispositivos conectados al sistema. El aplicativo en sitio se encargó de gestionar y ordenar, inclusive con perdidas de conexión, las mediciones y los mensajes de control. Por último, el servicio del backend se encargó de comunicar ambos productos de frontend y almacenar la información de los usuarios.

Teniendo en cuenta lo anterior, la arquitectura se diseñó bajo la posibilidad de soportar hasta 4000 dispositivos entre los cuales se encuentran computadores, celulares y cualquier dispositivo capaz de ejecutar rutinas en Python o Nodejs. Ademas, ambos aplicativos clientes fueron diseñados para ser multiplataforma. Por el alcance del proyecto no se realizaron pruebas a gran escala, es decir, con más de 4 dispositivos conectados.

*Palabras clave:* serverless, Json web token, comunicaciones móviles, comunicación industrial, control industrial, gestión de redes.

---

# **IMPLEMENTATION OF A RESOURCE MANAGER WITH ENVIRONMENTAL INTELLIGENCE FOR A SOLAR DECATHLON LATIN AMERICA 2019**

This document will explain the process carried out to implement a client server model that communicates different devices under the internet of things paradigm. The system consists mainly of 3 products: the backend product, a mobile application and a desktop application. The mobile application was, in broad strokes, responsible for remotely monitoring and controlling all the devices connected to the system. The application on site was responsible for managing and ordering, even with a lost connection, the measurements and control messages. Finally, the backend service was responsible for communicating both frontend products and storing user information.

Taking into account the above, the architecture was designed under the possibility of supporting up to 4000 devices among which are computers, cell phones and any device capable of running routines in Python or Nodejs. In addition, both client applications were designed to be cross-platform. Due to the scope of the project, no large-scale tests were performed, that is, with more than 4 devices connected.

*Keywords:* serverless, Json web token, mobile communication, industrial communication, industrial control, computer network management

## 1. Marco Teórico

Para la óptima comprensión de este proyecto de ingeniería es necesario el entendimiento de diferentes conceptos, tecnologías o teorías. Estos conceptos serán explicados de manera breve en el transcurso de este capítulo,

### 1.1. Arquitectura.

Se define como un conjunto de componentes de software que operan de manera relacionada y utilizan interfaces entre ellos. Los componentes de software utilizan un conjunto de herramientas de programación como APIs, Bibliotecas o Frameworks, para su óptimo diseño. Como aclaración las herramientas mencionadas anteriormente se definen como:

- **API (Application Programming Interface):** Es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta lenguaje de programación para ser utilizado por otro software para eliminar capas de complejidad.
- **Framework:** En el marco de desarrollo de software, un Framework (entorno de trabajo) corresponde a la estructura modular que sirve de base para la organización y desarrollo de software. Normalmente, pueden incluir soporte para diferentes lenguajes de programación, librerías, entre otras.
- **Biblioteca:** En programación, una biblioteca o librería es un conjunto de implementaciones funcionales, diseñadas en un lenguaje de programación específico, la cual ofrece una interfaz para la funcionalidad que se invoca.

El sistema que se implementó se basa en un modelo cliente servidor y consta de 3 bloques importantes; un servicio en la nube, una aplicación celular y una de escritorio. Estos a su vez utilizan herramientas, como las anteriormente descritas, para comunicar o relacionar elementos internamente.

### 1.2. Aplicación de escritorio.

Una aplicación de escritorio es cualquier software que pueda ser instalado en un computador o sistema de cómputo, y que permita ejecutar ciertas rutinas. En este sentido, una aplicación desarrollada como si fuese para web puede ser una aplicación de escritorio si se desarrolla utilizando el *framework* apropiado (Electron, Ionic). Los lenguajes de programación de aplicaciones de escritorio son un conjunto más amplio, los que más predominan son aquellos basados en máquinas virtuales o entornos de ejecución puesto que permiten realizar desarrollos independientes del sistema operativo. Dentro de los más conocidos están los lenguajes: Java, Python,

C++, Golang, etc.

Dentro de los elementos materiales del sistema se encuentran los dispositivos que van a encargarse de la capa física durante el proceso de comunicación y adquisición de datos, a continuación se definirán los componentes más importantes de estos componentes.

### **1.3. Aplicación web.**

Una aplicación web es un programa que se codifica en un lenguaje interpretable por los navegadores web, esto le permite a la aplicación ser independiente del sistema operativo y depender de interpretador web (navegador). Dentro de los lenguajes utilizados como desarrollo para aplicaciones web se tienen: HTML, JavaScript, Php, Asp, Python, Ruby, etc.

### **1.4. Servidor**

Desde el punto de vista del software, los servidores son programas de computador que atienden las peticiones de otros programas llamados clientes. Dentro de estos llamados los principales servicios de un servidor son: compartir datos, información y recursos de hardware. Desde el punto de vista de hardware toda la información entrante de los dispositivos (clientes) es recibida a través de una interfaz de red y reconocida para su posterior procesamiento y almacenamiento. Desde una perspectiva más simplista un servidor es un computador que ejecuta una serie de programas que le permiten ofrecer diferentes servicios a través de una red de computadores.

### **1.5. Tcp y Tls**

Tcp (Transmision control protocol) es un protocolo de comunicación de la capa de aplicación que está orientado la conexión, y según James F. Kurase, “Una conexión Tcp casi siempre es una conexión punto a punto, es decir, entre un único emisor y un único receptor”[3]. Por otra parte el protocolo Tls (Transport Layer Security) es una versión segura del protocolo Tcp y es una evolución del protocolo SSL (Secure Sockets Layer). Ambos protocolos son usados hoy en día por múltiples aplicaciones web, servicios bancarios y prácticamente cualquier cliente web. Sobre protocolos como estos se desarrollan las aplicaciones necesarias para crear servicios de programación de alto nivel como las interfaces con base de datos MySql y los servicios Http que utilizan las páginas web.

## 1.6. Jwt

Un Jwt (Json web token) es un estándar abierto (RFC-7519) basado en Json (JavaSript object notation) para crear un token que sirvan para enviar datos entre aplicaciones o servicios y garantizar que sean válidos y seguros. La información contenida en los tokens debe ser la necesaria para garantizar estos servicios de autenticación y caducidad de los tokens.

Todo JWT debe contener una cabecera que identifique el algoritmo de encriptación y un cuerpo que contenga un objeto Json con información necesaria para procesar la validez del token. Los posibles algoritmos de encriptación y los campos que puede soportar un JWT, están definidos en el estándar que se puede revisar en su portal oficial [4]

## 1.7. Mqtt

El transporte de telemetría para mensajes usando cola (Mqtt de las siglas en inglés: Message Queue Telemetry Transport) fue diseñada por IBM en 1999 para comunicación "machine to machine", con el objetivo de proporcionar un protocolo de mensajería de publicación-suscripción con los requisitos mínimos de ancho de banda, el tamaño de la huella del código, el consumo de energía y los datos de las cabeceras en general [5]. Este protocolo de comunicación está desarrollado sobre la capa de comunicación de Tcp o Tls para su versión segura.

El protocolo de comunicación Mqtt establece una comunicación bidireccional entre el broker y sus clientes. Un Bróker es un servidor que recibe todos los mensajes de los clientes y, en seguida, redirige estos mensajes a los clientes de destino relevantes; es decir, aquellos que estén suscritos a esa temática en especial. Un cliente es cualquier cosa que pueda interactuar con el bróker y recibir mensajes.

## 1.8. Http

Http es un protocolo de comunicación síncrono, es decir, el cliente espera a que el servidor responda. Los navegadores web tienen este requisito, pero el costo es la baja escalabilidad. Lo anterior, debido a que el cliente debe hacer una solicitud cada vez que necesite hacer una operación.

Http es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor pero es unidireccional, es decir que el cliente necesita iniciar la conexión. En un aplicativo de IOT (internet of things), los dispositivos y sensores generalmente son clientes, lo que significa que no pueden recibir comandos de la red de forma pasiva.

Http es un protocolo de uno a uno. El cliente realiza una solicitud y el servidor responde. Es difícil y costoso transmitir un mensaje a todos los dispositivos en red, lo que es algo común en aplicaciones de IoT.

### 1.9. Serverless

El concepto “serverless” define la capacidad de comprar una función como un servicio en el que el proveedor de la nube asume la responsabilidad de proporcionar un servidor y un entorno de ejecución bajo demanda para ejecutar el código.

Desde una perspectiva empresarial una empresa puede subcontratar todos los servicios asociados a los servidores y tener gastos únicamente por la cantidad de procesamiento, información o memoria RAM, utilizada. Como dijo Nate Taggart “Con serverless, pueden externalizar la necesidad del conjunto de habilidades de orquestación. Deje que Amazon, Microsoft o Google manejen la capa de orquestación. Deje que su equipo se centre en desarrollar aplicaciones y administrar el estado de las aplicaciones” [6].

Para finalizar, existe una discusión activa sobre lo que significa “serverless”, pero lo más acogido, es función como servicio [FaaS] y se funciona como un código que se desarrolla y se ejecuta bajo demanda en un proveedor de infraestructura específico.

### 1.10. Backend

El backend es la parte del código de una aplicación que corre en un servidor, es decir, es la capa de acceso a los datos almacenados de una aplicación; además, contiene la lógica de la aplicación que maneja dichos datos.

Los *backends* son los componentes funcionales de cualquier sistema orientado a servicios a través de protocolos estándar de Internet. Un *backend* tiene la característica de ser independiente de la plataforma que hace uso de sus servicios y del lenguaje de programación de la misma [7]. Lo anterior tiene que ver con que toda la interfaz de comunicación de un *backend* se desarrolla usando los protocolos estándar de internet.

Tradicionalmente el *backend* se comunica con el servicio de almacenamiento de datos (Mysql, Mongo Db, file system, etc) y utiliza lenguajes de programación como Ruby, Java, Python, Node.js, ASP, etc. para cumplir su propósitos de operación.

## 1.11. Frontend

El *frontend*, a diferencia del *backend*, es la parte de un aplicativo web que interactúa directamente con el usuario, por eso se dice que está del lado del cliente; es decir, desde el navegador o desde el dispositivo que utiliza el usuario final. En el *frontend* se encuentran todas las tecnologías de diseño y desarrollo web que corren en el navegador y que se encargan de la interactividad con los usuarios; también pueden ser tecnologías de escritorio o móvil pero lo más común es encontrar el desarrollo de *frontend* en la web.

## 1.12. Dispositivo

Un dispositivo es objeto hardware que puede ser visto como un sistema embebido con capacidades de procesamiento y comunicación a diferentes redes. En el dispositivo hardware reside el software que sirve como el manejador de las magnitudes físicas, voltajes, o sensores que están conectadas a él. Este componente normalmente posee las herramientas para el envío de información a servidores locales o remotos, en adición, dentro de los componentes que integran un dispositivo se tienen los siguientes:

- **Sistema de archivos:** Este componente de software es el encargado de almacenar los datos del sistema ya sea localmente, en un servidor de manera remota. Además de lo anterior, este módulo sirve para el almacenamiento de la información necesaria para el funcionamiento del sistema operativo.
- **Colector Cliente:** Este componente es el encargado de gestionar toda la información de los sensores que será enviada a manera de flujos de datos utilizando un protocolo de comunicación, esta capa de software añade las cabeceras y los formatos necesarias para la transmisión de datos.
- **Capa de Sensores:** Este software es el encargado de proveer todos los drivers lógicos para la conexión de los elementos físicos conectados al dispositivo.

Debido al tiempo estipulado para el proyecto, el dispositivo y los métodos de adquisición no serán de alta importancia en el desarrollo del sistema y se asumirá que todo desarrollo de los elementos de hardware debe estar solucionado junto con la comunicación con los mismos.

## 1.13. Teoría de colores

Desde el punto de vista del desarrollo de diseño gráfico es necesario conocer los conceptos que facilitan la implementación de una apropiada interfaz HMI. Por ejemplo la teoría del color es un grupo de reglas básicas en la mezcla de colores para conseguir el efecto emocional deseado combinando colores de luz o pigmentos.

Dentro de esta teoría existen varios modelos que explican este proceso: el modelo RGB, CYMK, YIQ, HSI etc.

Con el objetivo de comprender como se relacionan las emociones y los colores, se explicarán algunos términos comúnmente utilizados cuando se habla de colores y resultan necesarios para comprender el proceso de selección de la paleta de colores:

- **Armonías de color:** Los colores armónicos son aquellos que funcionan bien juntos, es decir, que producen un esquema de color sensible al mismo sentido (la armonía nace de la percepción de los sentidos y, a la vez, esta armonía retroalimenta al sentido, haciéndolo lograr el máximo equilibrio que es hacer sentir tensión o relajación).
- **Colores fríos:** En diseño, los colores fríos suelen usarse para dar sensación de tranquilidad, calma, seriedad y profesionalidad, también provocan la sensación de serenidad, recogimiento, la pasividad, el sentimentalismo, la sensación de frío. Como norma general son los colores que tienen azul y/o verde.
- **Colores cálidos:** En diseño, los colores cálidos son aquellos que están asociados a una sensación de alta temperatura. Como norma general, los colores cálidos son todos aquellos que van del rojo al amarillo, pasando por naranjas, marrones y dorados.
- **Colores complementarios:** Los colores complementarios son aquellos que se encuentran exactamente en el lugar opuesto del círculo cromático del modelo HSB. Es decir que cualquier color tiene su complemento a 180 grados de su valor HUE.

## 2. Introducción

### 2.1. Contexto introductorio

Durante este siglo, las nuevas generaciones que nacen rodeadas de la tecnología están cambiando las costumbres y el paradigma actual de lo que significa una vivienda. Con una sociedad cada vez más relacionada con la tecnología, nuevos desafíos de diseño e implementación se plantean para satisfacer las expectativas tecnológicas de las nuevas generaciones.

Básicamente el más importante beneficio utilizar la tecnología en viviendas comerciales es el proveer de servicios y facilidades a personas discapacitadas y ancianos [8], por otra parte, la adición de herramientas de medición y control permite ha permitido crear modelos de ciudad inteligente e interconectada. La popularidad de sistemas inteligentes ha venido incrementando debido al confort adicional, y herramientas de seguridad que pueden recibir los usuarios.

Por lo tanto, hay ciertos factores que deben ser tenidos en cuenta a la hora de diseñar un sistema de control y monitoreo de la vivienda, en este documento se explicará el proceso de diseño e implementación de un software para el manejo inteligente de una vivienda del Solar Decathlon Latinoamérica utilizando diferentes tecnologías de *frontend* y *backend*

La arquitectura desarrollada se puede describir en 3 productos:

1. Un servicio de *backend* serverless basado en: un broker de MQTT, un servicio de almacenamiento de datos tipo S3, una base de datos relacional tipo MySQL e interfaces HTTP basadas en funciones de nube.
2. Un programa encargado de administrar y controlar todas las rutinas de comunicación, reporte y almacenamiento en el sitio de control con su respectiva interfaz de usuario.
3. Una aplicación celular capaz de monitorear y controlar los datos generados localmente y los actuadores de la vivienda del Solar Decathlon.

### 2.2. Lineamientos de desarrollo

Debido a la naturaleza del proyecto, parte de los aspectos definidos al momento de establecer el anteproyecto fueron unas necesidades funcionales explicadas en el formato 1. Estas definieron el norte del desarrollo del proyecto.

	Universidad del Valle —Implementación de un gestor de recursos para el hogar con inteligencia ambiental.—	Rev: 003
Título: <b>ESPECIFICACIÓN DE REQUERIMIENTOS FUNCIONALES</b>	Documento : ERF-000	Página : 1 de 1

REVISIÓN HISTÓRICA			
Rev.	Descripción del Cambio	Autor	Fecha
001	Construcción del documento	Juan David Ramirez Villegas	18/03/2017
002	Correcciones	Juan David Ramírez Villegas	15/11/2017
003	Revisión	Juan David Ramírez Villegas	18/12/2017

Ref #	Funciones	Categoría
1.0	USER APP (application móvil)	O
1.1	El usuario debe poder acceder a la información medida en tiempo real.	E
1.2	El usuario debe poder acceder a los datos históricos recopilados en el mes y la relación de sus gastos con ellos.	O
1.3	El usuario debe poder cambiar los parámetros de configuración escogidos por defecto para la vivienda del Solar Decathlon.	O
1.4	El usuario debe recibir recomendaciones para mejorar su entorno y su huella hídrica cada cierto tiempo.	E
1.5	La aplicación notificará al usuario cuando la factura de energía o agua esté vencida.	O
1.6	El usuario debe ser capaz de ver un índice del impacto ambiental de su estilo de vida basado en datos cualitativos y cuantitativos	E

1.7	El sistema debe ser capaz de notificar las pérdidas eléctricas, y por consiguiente económicas, de un mal uso de los horarios establecidos por defecto.	O
2.0	HOUSE MANAGER (Aplicación en sitio)	E
2.1	El sistema debe mostrar gráficamente la cantidad de agua y potencia consumida durante el día contrastándolas con un valor máximo recomendado.	E
2.2	Por defecto, el uso de las cargas eléctricas más significativas debe programarse durante los picos de generación en la casa. Estos horarios podrán ser modificados bajo una advertencia de uso no eficiente.	E
2.3	El usuario debe poder acceder a la información medida en tiempo real.	E
2.4	El sistema debe poder comunicarse con una base de datos que represente todas las variables y el estado de los circuitos de la vivienda.	E

Tabla 1: Requerimientos Funcionales del sistema

La columna de la categoría indica si el requerimiento indicado es uno esencial (E) u opcional (O) para el desarrollo del sistema. Este documento será de gran importancia puesto que definió los lineamientos de desarrollo de todas las aplicaciones.

## **3. Diseño del sistema**

### **3.1. Diseño conceptual**

Como primera medida se decidió diseñar el sistema con 2 programas (escritorio y celular) que fueran capaces de acceder a la base de datos directamente, porque resultaba ser la topología más sencilla para lograr los requerimientos funcionales. El diagrama de bloques del concepto se puede observar en la figura 1. Teniendo en cuenta la figura 1, los bloques azules representan objetos de importancia para la comprensión del flujo de la información en el sistema, los bloques grises representan rutinas importantes de los objetos anteriormente mencionados y los bloques naranjas son directamente la interfaz de comunicación con el servidor. En el diagrama se puede observar como se planeó tener la comunicación con el servidor MySQL directamente a través de sus APIs. También, se puede ver como la relación Raspberry y celular es de generador y visualizador, es decir, que con esta topología no se podía controlar la vivienda de manera remota.

Durante el proceso de implementación se cambió el diseño a una topología con más seguridad y escalabilidad, esto debido a que la anterior topología permitía el acceso directo de los programas cliente a la base de datos, exponiéndola a brechas de seguridad.

### **3.2. Diseño final**

El diseño final contempla las mejores condiciones de: modularidad, costo, tamaño y escalabilidad. Para cada uno de estas características se escogieron tecnologías y se desarrollaron funciones claves.

Desde la perspectiva de la modularidad, se diseñaron aplicaciones de *frontend* capaces de integrar 2 tipos de protocolos de comunicación: MQTT y HTTPS; además, para acceder a los servicios de la nube se seleccionó el proveedor de servicios de backend “Google Cloud Platform” puesto que permitirá utilizar de la mejor manera todos los servicios de Google: su asistente de voz, su interfaz de inteligencia artificial, sus servicios de autenticación, su red privada de fibra óptica (que es de las más rápidas y de mayor cobertura), entre otros. Una comparación más a fondo de sus competidores y los motivos de esta selección se puede observar en el capítulo “Serverless backend”.

En cuanto al costo, se utilizaron tecnologías de *backend* totalmente escalables tipo *serverless* para ajustar los gastos a únicamente los necesarios, teniendo en cuenta que las pruebas a gran escala estuvieron lejos del alcance de este proyecto, aún así se

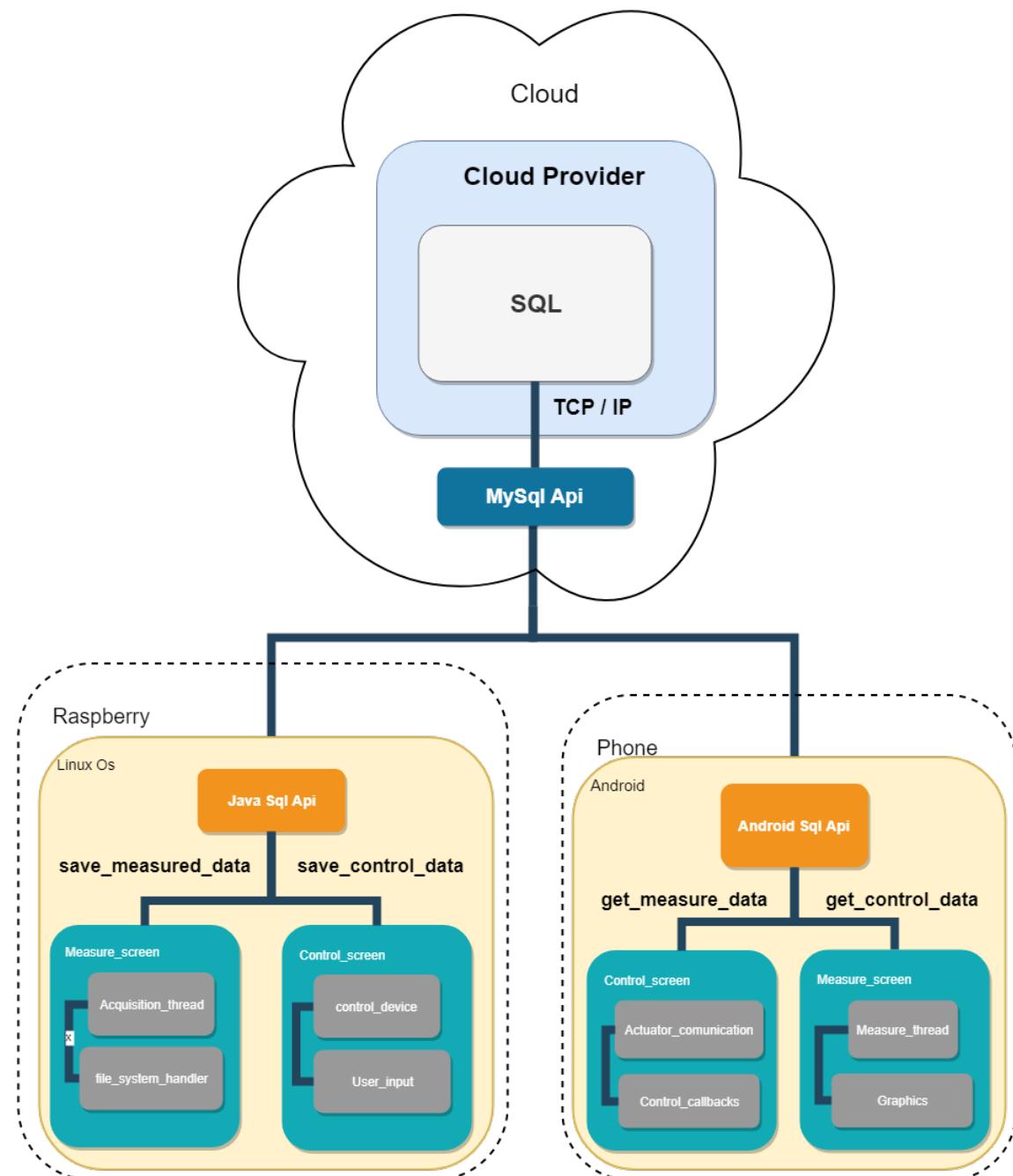


Figura 1: Primer diseño generalizado de la estructura del sistema Fuente: propia

desarrolló toda la arquitectura pensando en la necesidad de optimizar en la mayor medida los gastos. Teniendo en cuenta que no se hicieron pruebas a gran escala los servicios de *backend* fueron posibles gracias a las cuotas gratuitas del mayorista utilizado para desplegar el servicio (Google).

En cuanto al requerimiento de escalabilidad, la decisión de diseñar serverless jugó el papel más importante: Los servicios servirles le permitieron a la aplicación ser diseñada para permanecer funcional sin hacer ningún cambio inclusive con 4000 dispositivos conectados (esta última apreciación es teórica puesto que no se hicieron pruebas a esa escala).

El producto del *backend* se diseño para utilizar el servicio de Google Cloud y utilizó los siguientes elementos para su funcionamiento: Un “Google Cloud Storage Bucket” conectado a la red privada de Google para el almacenamiento de archivos, Una base de datos relacional MySql para el almacenamiento de datos de monitoreo y control de los dispositivos, una lista de “funciones de nube” activadas por solicitudes Https y por ellas mismas que se comunican directamente con la base de datos y finalmente, un *broker* nativo de Google Cloud llamado “Cloud IOT” que recibe las conexiones de Mqtts de los dispositivos del sistema.

La aplicación móvil se diseñó para estar compuesta de 3 escenas: la escena de control, utilizada para el manejo de los actuadores disponibles desde el aplicativo en sitio; la escena de medición, encargada de presentarle al usuario las mediciones adquiridas por el aplicativo en sitio en tiempo real; y la escena principal, donde se puede observar un indicador del porcentaje de las variables mas relevantes respecto a un punto de referencia. Por ejemplo, el porcentaje de consumo de agua con respecto al promedio de consumo de agua en un mes de un grupo familiar.

Finalmente, el aplicativo en sitio se diseñó para manejar un flujo de datos más grandes y la posibilidad de funcionar desconectado de la red. La aplicación se diseño para permitirle al usuario programar rutinas de control de los circuitos a partir de horarios, visualizar los datos medidos, cambiar parámetros de adquisición como el tamaño del *buffer* y la frecuencia de muestreo, entre otros.

Para comprender mejor el diseño, se puede observar en la figura 2 un diagrama generalizado de los componentes de software del sistema, donde principalmente los desarrollos hacen parte de 3 grandes bloques: el bloque de la nube, el bloque del dispositivo embebido (Raspberry), y el bloque del dispositivo móvil (Android).

En el nuevo diseño de programa se puede observar como la comunicación con el *backend* esta gobernada por el protocolo Mqtts desde la perspectiva de la aplicación de escritorio; para esto se utiliza el cliente de Mqtts de Python. Por otro lado, la co-

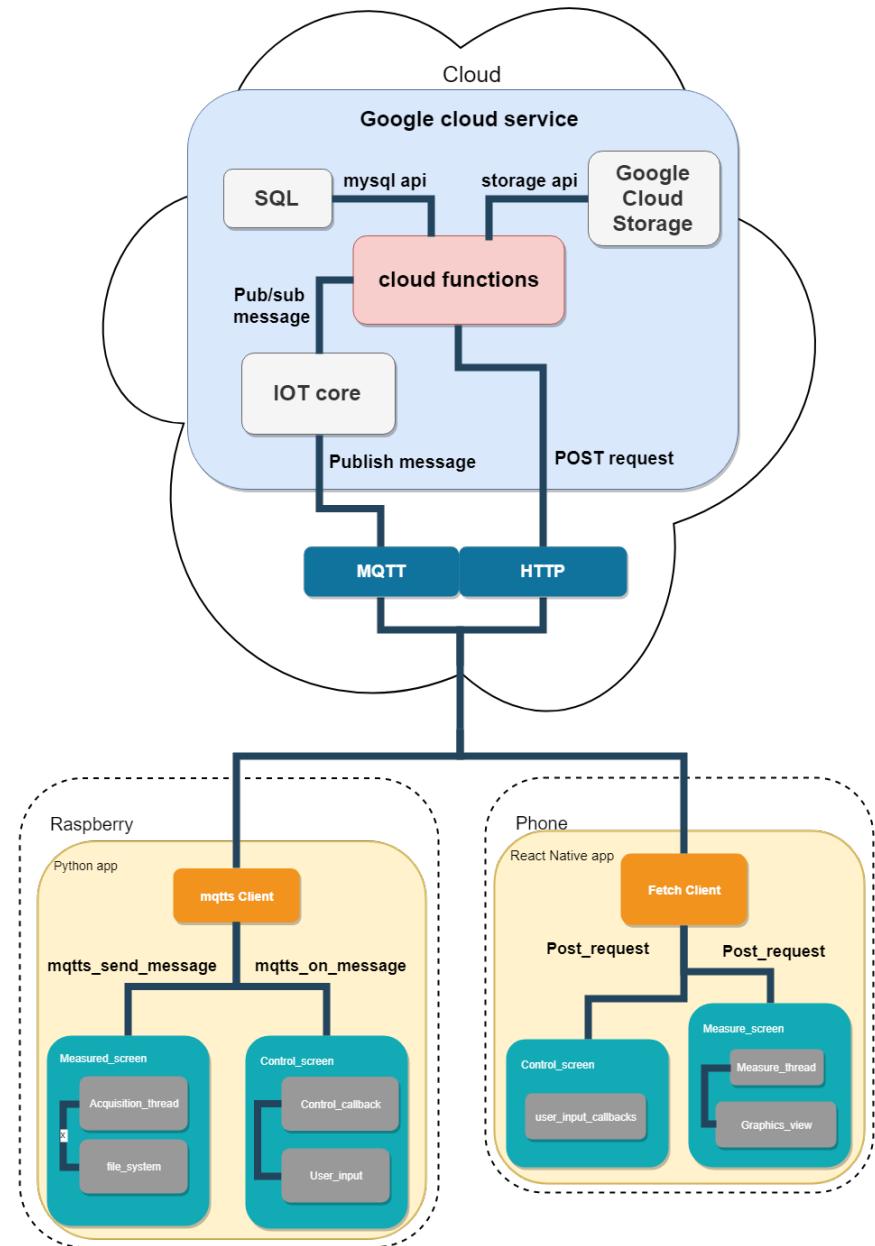


Figura 2: Diagrama generalizado de las entidades software del proyecto. Fuente: propia

municación del cliente móvil esta gobernada por el cliente Fetch de Javascript. Otro cambio importante del diseño del sistema, es que en la ventana de control se eliminó el componente de visualización del estado actual de control, y se cambio por un generador de mensajes de control a través de solicitudes Https tipo Post. Teniendo en cuenta lo anterior, se añadió una rutina de atención de eventos de control, provenientes del *backend*. Es decir que, ahora la aplicación de escritorio podría recibir órdenes de control desde la aplicación celular.

El flujo de la información del diagrama generalizado se puede observar en el diagrama de secuencia de la figura 3. En el diagrama se observan los eventos y las funciones más importantes involucradas en el proceso de comunicación entre aplicaciones.

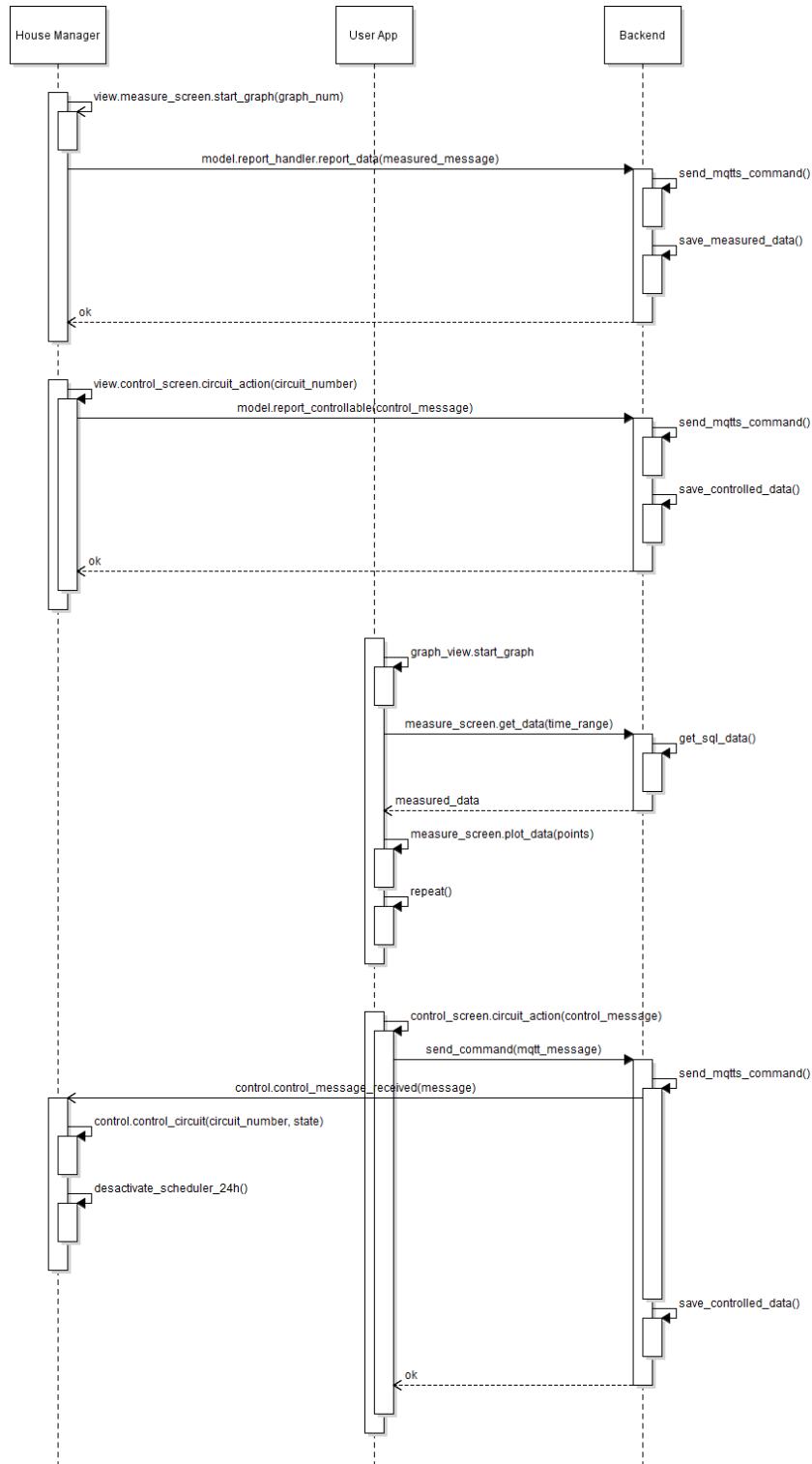


Figura 3: Diagrama de secuencia general de la comunicación. Fuente: propia

## 4. Api de animación

### 4.1. Investigación y conceptualización

Después de imaginar y delimitar el problema a tratar, como primera instancia, se seleccionó el tipo de entorno de trabajo que sería utilizado para el desarrollo del aplicativo ejecutable de manera local en la vivienda del Solar Decathlon. Durante el transcurso del documento este programa será llamado “House Manager”.

Para seleccionar el entorno de trabajo se realizó una averiguación de los posibles entornos de desarrollo que permitieran implementar programación orientada a objetos en dispositivos embebidos, dentro de los entornos de desarrollo más conocidos se observó: Python, Java, Procesing, C++, entre otros. Todos los anteriores con comunidades muy enfocadas a su mejoramiento constante. Teniendo en cuenta lo anterior realizo un análisis por separado como se puede observar a continuación:

- En el caso de C++ se observó que el entorno de trabajo más apropiado para desarrollar aplicativos gráficos era .net, la desventaja de este aplicativo era su estrecha relación con el lenguaje de programación de bajo nivel c#, por ende el manejo de memoria del programa incrementa en gran medida los costos de desarrollo.
- Para Python se observó que los entornos de desarrollo de interfaz gráfica nativos tenían apariencias precarias similar a la nativa de Windows 98, en adición, los frameworks más estéticos tenían un soporte independiente al del lenguaje de programación como tal, por otra parte el lenguaje de programación como tal era multiplataforma y de muy alto nivel, lo que le permitiría al programador desarrollar algoritmos de mayor funcionalidad con menos líneas de código.
- En el caso de Java se observó que el entorno de desarrollo para interfaces gráficas era nativo del lenguaje de programación y permitía una mayor flexibilidad a la hora de desarrollar esquemas complejos con la excepción que se debía codificar detalladamente el comportamiento del objeto, lo que significaba más tiempo de trabajo para el programador y archivos con más líneas de código.

Como etapa de conceptualización, se realizó una primera versión el diseño del aplicativo principal junto con sus respectivas escenas (Fig 4). Los elementos utilizados en esta primera etapa de diseño se consistieron principalmente de: botones, caja de opciones, contenedores, y paneles animados. Para esta etapa de conceptualización no se tenía una paleta de colores escogida pero si se buscaba tener en cuenta tonalidades verdes siendo congruentes con la temática amigable con el medio ambiente que rodea al Solar Decathlon, también, se persiguió el poder seleccionar a partir de la pantalla principal diferentes “sub-aplicativos” que podrían ser diseñados de manera independiente.

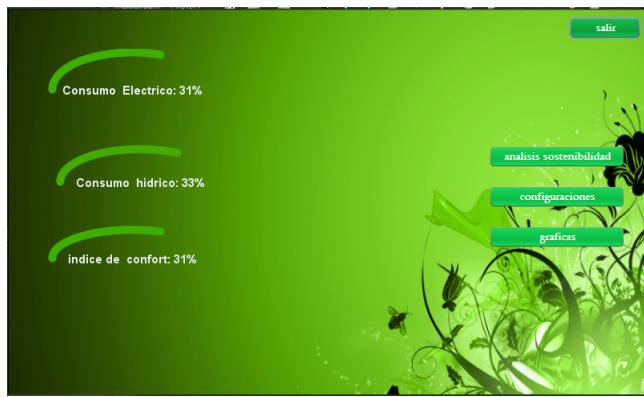


Figura 4: Conceptualización inicial del api en sitio. Fuente: propia

Para hacer realidad el diseño de esta primera etapa de conceptualización se utilizó como lenguaje de programación el lenguaje de programación Java puesto que representaba en su mayor parte el lenguaje más utilizado en la Universidad del Valle durante el momento.

Además, Java es un lenguaje de programación de muy alta demanda, de hecho, el de mayor ofertas de trabajo registradas durante el 2017 en el portal de trabajo internacional indeed.com, seguido de JavaScript, c# y Python, tal como se observa en la figura 5. Pero esta característica se debe principalmente a que el sistema bancario tradicional esta construido sobre servicios de backend utilizando Java.

## 4.2. Api de java

Todos los objetos implementados para la api fueron desarrollados estrictamente con java puro y sin librerías externas con el objetivo de obtener un código limpio, totalmente multiplataforma y que heredara paquetes únicamente de los módulos *Swing* nativos de java. Como primera instancia se desarrolló el componente funcional de la API, es decir la parte programática, y finalmente se escribió el Javadoc, el cual es un documento integrado que describe como se deben usar los objetos

Normalmente para java es necesario construir y empaquetar el código que contenga la interacción de manera manual con java puro. Por ejemplo, en JavaScript es posible enlazar funciones directamente sin acceder a paquetes adicionales como los “event listeners” de java, puesto que todo es considerado un objeto; desde un numero hasta una función.

El primer objeto desarrollado en la Api de java fue el botón personalizado, usando este objeto contenido en la api se puede crear un botón de cualquier tamaño con una, dos o tres imágenes para los estados; “normal”, “presionado” y “encima”. Un ejemplo de la utilización de la api se puede ver en la figura 6. Este objeto funciona

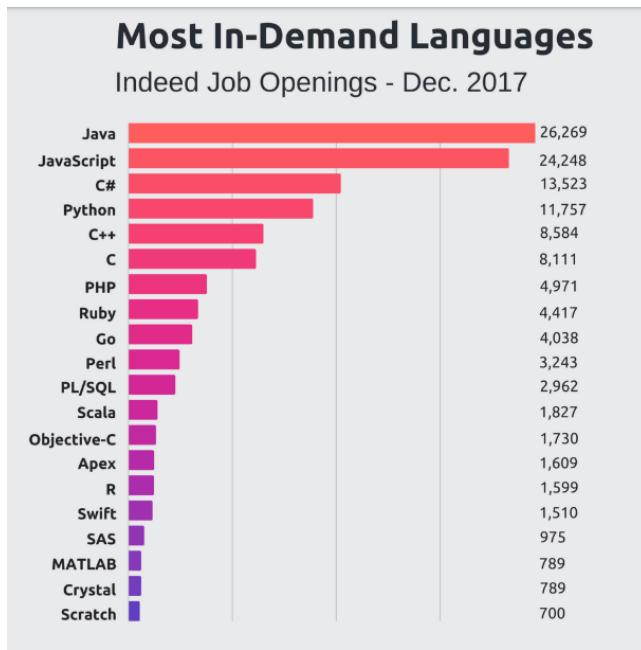


Figura 5: Estadísticas de los lenguajes de programación con más demanda. Fuente: [1]

ajustando automáticamente su tamaño basado en la resolución de las imágenes, esto para complementar con una metodología de diseño: primero el dibujo, luego el programa.

El segundo objeto gráfico desarrollado fue un botón expandible o “caja de opciones”, que permitiera generar un botón desplegable con la posibilidad de modificar texto de manera superficial basado en dos o tres imágenes para la parte superior, inferior y media respectivamente. Una implementación utilizada en la aplicación se puede observar en la figura 7. Cada una de las imágenes del botón desplegable debe tener sus 3 imágenes de estado: normal, presionado y encima para interactuar de manera dinámica con el mouse.

Como tercer objeto gráfico, se desarrolló un objeto de animación que podía mover un objeto tipo swing (esta es la clase madre de la mayoría de los objetos gráficos en java) en 4 posibles direcciones; las dos horizontales y las dos verticales, tal como se observa en la figura 8. La función permite modificar la ubicación actual del objeto moviéndolo de manera lineal con una tasa de refresco dada (actualización visual) y una velocidad constante, desde una coordenada X o Y dependiendo de en cual eje coordinado se deseé mover el componente.

El desarrollo de la api fue de importancia para la construcción de la primera versión del aplicativo en sitio, la primera versión se puede observar en la figura 9, debido a la recursiva implementación gráfica se obtuvo un programa que sobre-cargaba un sistema de cómputo completo (portátil Dell con procesador i5 segunda

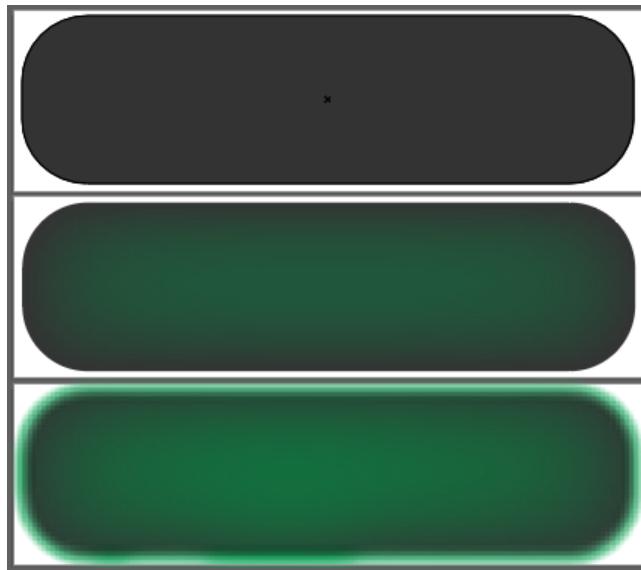


Figura 6: El diseño de estados del botón interactivo. Fuente: propia

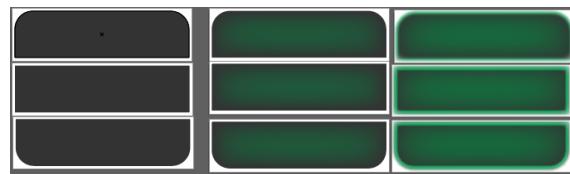


Figura 7: El diseño de los estados del botón interactivo desplegable. Fuente: propia



Figura 8: Movimiento según la ubicación de la función horizontal. Fuente: propia

generación, 4gb de ram DDR3, y un disco duro HDD). Por lo tanto se buscó cambiar el entorno de desarrollo a uno con más flexibilidad a la hora del diseño gráfico, con capacidad de funcionar en sistemas de más bajo cómputo tipo Single Computer Boards como la Raspberry o la Beaglebone. Para probar el desempeño de la api, se realizaron pruebas con los elementos ya diseñados en la api de Java y se observó que resultaba una carga para el cpu realizar las animaciones, principalmente, cuando se deseaba mover más de 5 objetos tipo JavaSwing (los objetos gráficos nativos de java).

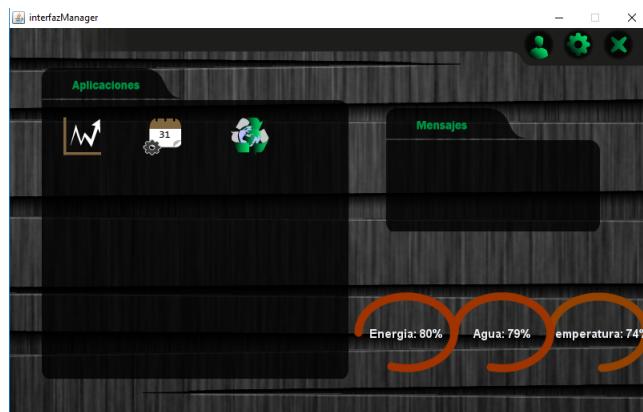


Figura 9: Primer Aproximación del diseño de la interfaz en sitio. Fuente: propia

Después de haber programado los elementos necesarios para generar la estructura básica del aplicativo se replanteo el diseño con una estructura más moderna y parecidas a las tendencias en diseño móvil tal como se observa en la figura 10, con el objetivo de que se percibiera que ambos aplicativos desarrollados en este sistema podían ofrecer los mismos servicios. Además, generar una experiencia de usuario similar en ambas plataformas



Figura 10: Ajuste de diseño de la interfaz en sitio. Fuente: propia

En un tercer nivel de conceptualización se re diseño la interfaz gráfica estandarizando y simplificando los componentes que esta contenía; se seleccionó la paleta

de colores y agregaron algunos nuevos objetos al diseño generalizado como se puede observar en la figura véase en la figura 11. Para la correcta selección de la paleta de colores se buscó generar una simetría triangular en el espacio de colores HSV, usando como referencia un color sacado del logo oficial de la “Chameleon House” (nombre oficial de la casa diseñada para el Solar Decathlon).

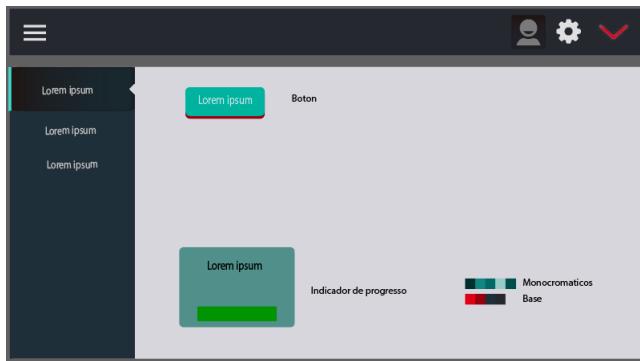


Figura 11: Versión final del concepto de diseño gráfico. Fuente: propia

Debido a la estructura de Java, codificar todos los nuevos objetos resultaba una tarea de mayor trabajo y con baja recompensa, es decir, más líneas de código por objeto. Como solución, se tradujo lo que se tenía desarrollado hasta el momento, a un lenguaje de programación de más alto nivel (Python) y se utilizó un framework exclusivamente para el desarrollo de la interfaz gráfica. Debido a que el framework se trataba de un módulo completamente escrito en C, le permitía a la aplicación tener un mejor rendimiento gráfico.

En adición, se cambió el lenguaje de programación para optimizar el tiempo de desarrollo de la aplicación. Teniendo en cuenta que python es un lenguaje de mas alto nivel se decidió cambiar de lenguaje para optimizar el tiempo desarrollo de la aplicación en sitio. Para ilustrar lo anterior se puede observar un snippet<sup>1</sup> de ejemplo para java y python respectivamente:

```
public static void POSTRequest() throws IOException {
    final String POST_PARAMS = "";
    System.out.println(POST_PARAMS);
    URL obj = new URL("https://jsonplaceholder.typicode.com/posts");
    HttpURLConnection postConnection = (HttpURLConnection) obj.openConnection();
    postConnection.setRequestMethod("POST");
```

---

<sup>1</sup>En programación un “snippet” es un término para definir una pequeña porción de código reusable. Usualmente son unidades operativas que son incorporadas en modulos de programa mas grandes.

```
postConnection.setRequestProperty("userId", "a1bcdefgh");
postConnection.setRequestProperty("Content-Type", "application/json");
postConnection.setDoOutput(true);
OutputStream os = postConnection.getOutputStream();
os.write(POST_PARAMS.getBytes());
os.flush();
os.close();
int responseCode = postConnection.getResponseCode();
System.out.println("POST Response Code : " + responseCode);
System.out.println("POST Response Message : " + postConnection.getResponseMessage());
if (responseCode == HttpURLConnection.HTTP_CREATED) { //success
    BufferedReader in = new BufferedReader(new InputStreamReader(
        postConnection.getInputStream()));
    String inputLine;
    StringBuffer response = new StringBuffer();
    while ((inputLine = in.readLine()) != null) {
        response.append(inputLine);
    }
    in.close();
    // print result
    System.out.println(response.toString());
}
}
```

El “snippet” anterior es utilizado para realizar una solicitud Https tipo post y utiliza 28 líneas de código. Por el contrario, en Python la misma solicitud se puede realizar en 4 líneas de código, como se observa a continuación:

```
response = req.post('https://jsonplaceholder.typicode.com/posts', data = None,
if response != None:
    print(response.json())
```

Una ventaja de utilizar java como lenguaje de programación es que garantiza de manera efectiva la estructura de los datos durante la ejecución, pero sacrifica tiempo de desarrollo. El anterior ejemplo fue utilizado debido a que la solicitud Https tipo Post es utilizada por los programas cliente del proyecto. Teniendo en cuenta lo anterior, se decidió cambiar el lenguaje de desarrollo a Python inclusive con la Api de desarrollo gráfico terminada.

### 4.3. Api de Python y Qt

El framework utilizado para desarrollar los módulos gráficos se llama Qt y para poder utilizarlo es necesario instalar unas dependencias C específicas y la librería

respectiva para Python. Una vez Qt está correctamente configurado el diseño estático del aplicativo se realiza a partir de un “lenguaje de marcado” (markup language) auto generado llamado Qml que define las propiedades gráficas estáticas de los objetos a utilizar, el Qml viene de la mano con un lenguaje tipo “hoja de estilo” (style sheet) llamado Qss, a partir del cual se le adjudican propiedades interactivas y estilos que en conjunto son comprendidos por Python y su librería Pyside. El anterior entorno de desarrollo, facilitó la implementación de la api puesto que se relacionaba directamente con la forma como se desarrolla frontend para la web. Para esta etapa de diseño se utilizó una paleta de colores complementarios para los objetos importantes y tonalidades oscuras de grises azulados para los fondos. Todos dentro del espectro de colores de la marca de la casa del Solar Decathlon Latinoamerica de la Universidad del Valle.

Teniendo en cuenta el nuevo framework se utilizó la herramienta de diseño de Qt desing (oficial del framework) para codificar de manera automática las propiedades estáticas de la interfaz gráfica. También, se codificaron manualmente las propiedades dinámicas en un archivo de Qss que luego fue cargado por la librería Pyside para ser procesado como un objeto Python.

Primero, se codificó un botón de la misma manera como se implemento el código de la API para Java; con este objeto de Python se logró crear un botón de cualquier tamaño con una, dos o tres imágenes para los estados; “normal”, “presionado” y “encima”. Un ejemplo de la utilización de la api se puede ver en la figura 12.



Figura 12: Botón de tamaño flexible genérico de la api de qt. Fuente: propia

Segundo se codificó un botón genérico con las mismas funcionalidades del botón anterior, pero en lugar de funcionar basado en imágenes funcionaba con un color dado y las interacciones “normal”, “presionado” y “encima” son calculadas atenuando y resaltando el color para generar un efecto de sombra e iluminación. Para el aplicativo se utilizó el color mostrado en la figura 13.

Tercero, se desarrolló un expandible o “caja de opciones”, personalizada, para ello

se modificó el objeto de Qss “QOptionbar” que permite generar un botón desplegable con la posibilidad de modificar texto de manera superficial basado en un color específico.

Cuarto, se programó una barra deslizable para facilitar el proceso de ingresar datos numéricos a la aplicación desde una interfaz táctil. Nuevamente se usaron los colores de la paleta de colores seleccionada y se optó por la solución de diseño más sencilla posible tal como se puede ver en la imagen 13.

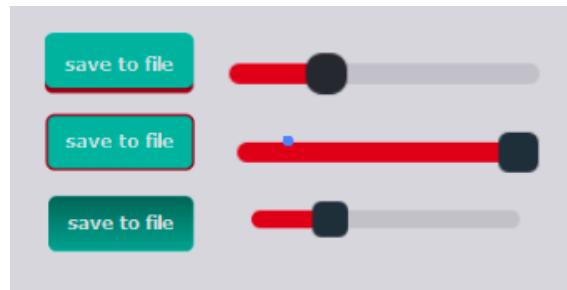


Figura 13: Boton de tamaño fijo usando imágenes y slider. Fuente: propia

Finalmente se personalizaron los colores de todos los espacios y elementos secundarios utilizados, como: la gráfica, la barra de menú, los botones especiales del menú, etc. Con un total de 6 elementos diseñados e implementados capaces de reutilizarse para otras interfaces.

## 5. House Manager

Una vez descritos todos los componentes gráficos necesarios para realizar la aplicación, se implementaron los objetos de modelo, vista y controlador, basándose en los requerimientos descritos en la hoja “requerimientos funcionales” de la tabla 1. Para el desarrollo de estos componentes fueron necesarias diferentes condiciones técnicas que serán descritas en el siguiente apartado.

### 5.1. Especificaciones técnicas

La aplicación fue desarrollada utilizando el paradigma MVC, para ello, se organizó el proyecto en 3 carpetas: *view*, *control* y *model*, donde todos los componentes y *scripts* son gestionados desde un archivo principal. Debido a la naturaleza gráfica del proyecto y las limitaciones técnicas del hardware raspberry, se utilizó Python 2.7 para garantizar la compatibilidad de la mayor cantidad de sistemas operativos y librerías. Para el desarrollo de la aplicación fueron utilizadas las siguientes librerías:

- **Paho-mqtt:** Esta librería se utilizó para establecer la comunicación con el bróker de Mqtt nativo de “Google Cloud Iot”.
- **Numpy, Scipy:** Estas listreries fueron utilizadas para facilitar el procesamiento de vectores y mejorar el tiempo de procesamiento de los mismos.
- **Pyqtgraph:** Fue utilizada para gestionar el renderizado de las gráficas con el mejor valor posible de fotogramas por segundo. La necesidad de optimizar el componente grafico fue de importancia puesto que se desarrollaría sobre una tarjeta de computación de propósito específico (Raspberry).
- : **Http.client:** Esta librería fue utilizada para establecer una comunicación basada en solicitudes http con un “endpoint” del backend.
- : **Pyjwt, Cryptography:** Con esta librería se desarrolló la etapa de encriptación de datos adicional basada en *Java Web Tokens*.

Pensando en la escalabilidad del sistema, se definió un espacio para las “aplicaciones” adicionales donde se pudieran incluir nuevas rutinas de adquisición y control para dispositivos nuevos 14. Como prueba de concepto se utilizó una rutina serial para adquirir el valor de khw acumulado en un contador eléctrico bifásico Inelca. Respecto a esta consideración de diseño se hablará más en el capítulo de “pruebas y resultados”.

En adición, se diagramó la experiencia de usuario de tal forma que el control y la medición de los dispositivos conectados a la tarjeta tuvieran el mismo “costo

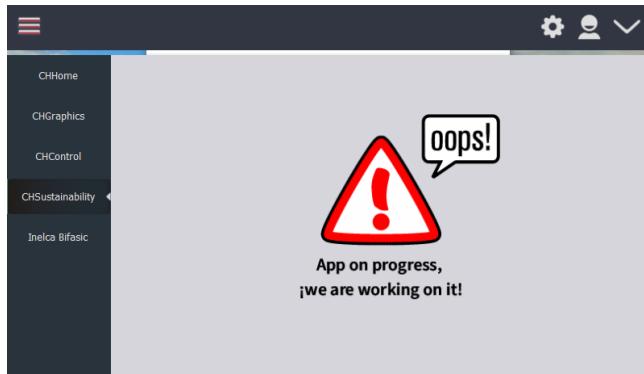


Figura 14: Visualización de la escalabilidad del sistema con una App no desarrollada por completo. Fuente: propia

de interacción”, es decir, que ambos estuvieran a la misma cantidad de clicks desde cualquier ventana del sistema. Por ende, todas las aplicaciones adicionadas en versiones futuras serán más fáciles de asimilar para el usuario.

### 5.2. Requerimientos Funcionales

Durante la etapa de conceptualización del proyecto de ingeniería que precede este documento se establecieron unos requerimientos funcionales con el fin de guiar el proceso de desarrollo. Este documento se puede ver en la tabla 1, y la implementación de cada requerimiento se puede observar a continuación:

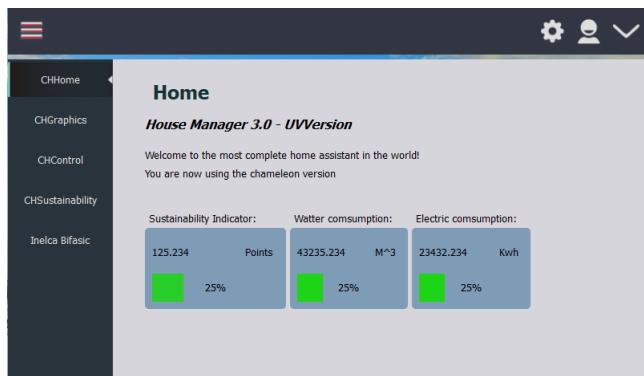


Figura 15: Escena del home del House Manager con barras de progreso para la indicación dinámica. Fuente: propia

1. “El sistema debe mostrar gráficamente la cantidad de agua y potencia consumida durante el día contrastándolas con un valor máximo recomendado”: Para el desarrollo de esta funcionalidad se utilizó un componente gráfico tipo barra de progreso donde su 100 por ciento tenía como referencia la cantidad de

agua recomendada para el número de personas presentes en la vivienda<sup>15</sup>. El número de personas presentes en la vivienda fue un dato ingresado manualmente por configuración inicial (sobre el sistema de configuración se hablará en los siguientes apartados de este capítulo).

2. “*Por defecto el uso de las cargas eléctricas más significativas debe programarse durante los picos de generación en la casa, estos horarios podrán ser modificados bajo una advertencia de uso no eficiente*”: En miras de implementar esta funcionalidad se utilizó un elemento gráfico tipo calendario para configurar de manera individual las ventanas de activación de las cargas eléctricas en la vivienda<sup>16</sup>.

También, se consideró la opción de activación o desactivación inmediata de los circuitos; para ello se, implementó una rutina flexible que permitió ignorar las cargas eléctricas que el usuario había modificado manualmente (Esta rutina se nombró “Scheduler Handler”), lo anterior únicamente durante 24 horas para no afectar el objetivo de sostenibilidad de la vivienda.

Además, se utilizó la conexión Mqtt para accionar remotamente cualquiera de los circuitos localmente gestionados, en caso de recibir un mensaje de control para los circuitos el sistema también suspendiera por 24 horas la acción del “Schedule Handler”.

3. : “*El usuario debe poder acceder a la información medida en tiempo real*”: Para la implementación de este requerimiento se creo una ventana gráfica con la capacidad de mostrar el valor de la medida del sensor con un tiempo de muestreo entre 1 y 50 Hertz. Teniendo en cuenta lo anterior, el medidor debe ser capaz de almacenar el valor de consumo eléctrico y de agua internamente. Tal como lo hace el contador eléctrico bifásico de Inelca con el que se realizaron las pruebas.

Si el usuario lo desea, puede cambiar el numero de puntos y la frecuencia de muestreo de manera inmediata, moviendo las barras deslizables. Ademas, puede guardar los datos mostrados en la gráfica en el formato de excel presionando el botón “save to file”.

4. : “*El sistema debe poder comunicarse con una base de datos que represente todas las variables y el estado de los circuitos de la vivienda*”: Para la implementación de este requerimiento se creo un “endpoint” en el servicio de *backend* con la capacidad de hacer consultas de históricos a la base de datos (Un “endpoint” es un url web gestionado por un backend en el cual se pueden realizar solicitudes Http). La solución tiene este comportamiento puesto que la aplicación del cliente final no debe tener ningún tipo de credencial para el acceso a

la información directamente, con una capa de procesamiento se pueden integrar este tipo de solicitudes con el mismo sistema de encriptación del broker de Mqtts (para más información de los endpoints y la rutina de encriptación véase el capítulo *backend*).

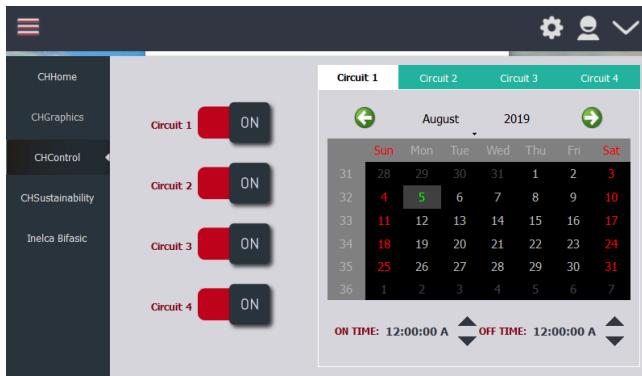


Figura 16: Escena de control del House Manager con sus calendarios. Fuente: propia

### 5.3. Funciones adicionales

En adición a los requerimientos funcionales, se añadieron funciones a la conceptualización de diseño original para mejorar el funcionamiento de la plataforma y la experiencia de usuario. Dentro de estas funciones podemos encontrar la configuración del sistema, la posibilidad de guardar localmente para no perder la información adquirida, etc. Para comprender mejor estas funciones adicionales, serán descritas a continuación:

- 1. Handler remoto:** Esta característica le permite al sistema cambiar de manera automática entre el modo remoto y el modo local para no perder información en ningún momento del reporte de datos. En caso de la caída repentina de la red, el programa es capaz de cambiar al modo local sin la perdida de ningún dato. También, es capaz de entender cuando el usuario desea que el dispositivo funcione permanentemente de manera local o cuando el sistema vuelve a estar nuevamente conectado a la red. Los datos que son guardados de manera local se almacenan en la carpeta personal del usuario, en archivos de excel, y son creados por día con la fecha como nombre. Teniendo en cuenta lo anterior, el nivel de autonomía local depende exclusivamente de la memoria en el disco duro; para la Raspberry fueron aproximadamente 10 GB de las 16 disponibles en la memoria micro SD.

2. **Sistema de almacenamiento local:** Con esta característica el House Manager puede almacenar todos los datos adquiridos en una hoja de Excel generando archivos organizados por días. También es capaz de almacenar en un archivo diferente los datos presentes en la gráficas teniendo en cuenta el día de adquisición. 17.



Figura 17: Escena de medición del House Manager, con su respectivo panel para guardar los datos. Fuente: propia

3. **Ajustes de configuración** Esta es una característica que incluye una ventana adicional donde se configuran los aspectos concernientes al sistema en general<sup>18</sup>. Los aspectos configurables del sistema son: La frecuencia en segundos con la que se reportan los datos al servidor, el costo del kilo watt hora, el costo del metro cúbico de agua, el día de notificación del correo (ver función “Notificación automática”), el modo de funcionamiento (remoto o local) y el nombre de la ubicación (este nombre es opcional y sirve para administrar los diferentes dispositivos desde la perspectiva del backend).

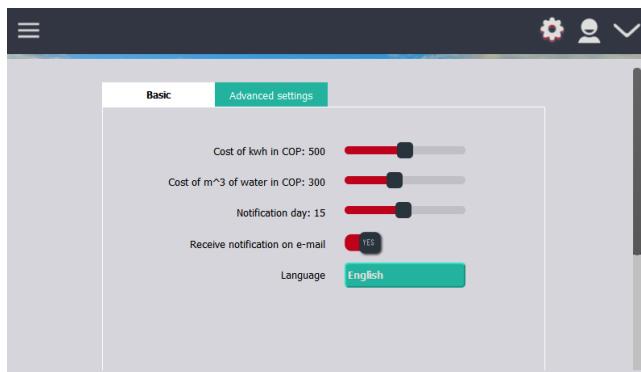


Figura 18: Escena de configuración del dispositivo House Manager. Fuente: propia

4. **Notificación Automática:** Para el correcto desarrollo de esta función se compró el dominio www.alfagenos.com para recibir las notificaciones desde el remitente de correo: notificaciones@alfagenos.com. Esta notificación tendrá lugar dependiendo de la configuración establecida e indicara el gasto en pesos del consumo eléctrico y de agua hasta la fecha establecida en la configuración. También notificará los gastos energéticos en horarios adicionales al pre establecido por la configuración del Solar Decathlon.
5. **Autenticación usando google:** Para desarrollar esta funcionalidad se utilizó la api de autenticación de google y se creo una escena aparte para la visualización del usuario 19.

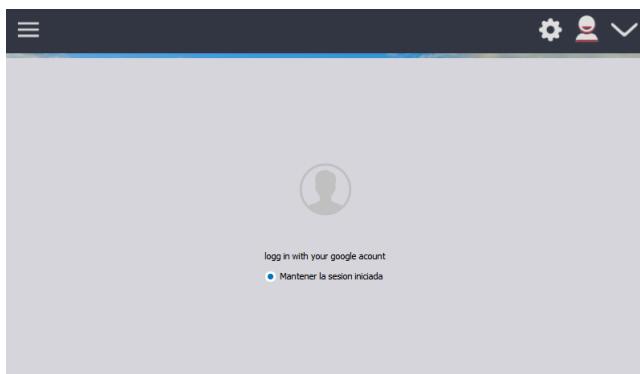


Figura 19: Escena para el inicio de sesión del House Manager. Fuente: propia

6. **Doble nivel de encriptación:** Debido a las exigencias de los servidores de Google se implementaron 2 capas de seguridad de encriptación para la comunicación de la aplicación con el backend. La primera capa es de tipo Tls y opera para cualquier producto de google, inclusive las operaciones realizadas por los navegadores. La segunda se desarrolló usando encriptación “RS256” a través de Jwt.

#### 5.4. Algoritmos Utilizados

Para cumplir con algunos requerimientos funcionales y algunas funciones adicionales, fue implementar hilos de proceso en segundo plano que se encargaron de tareas que no eran sensibles a pequeños cambios en la frecuencia de ejecución. Los algoritmos utilizados en estos hilos de programa, comparten la información de toda la aplicación a través de los 3 objetos principales: el controlador la vista y el modelo. Estos algoritmos o programas ocurren en segundo plano y no representan una carga significativa para el CPU ni se comportan de manera determinista respecto a los tiempos de ejecución establecidos para los mismos.

#### 5.4.1. Scheduler Handler

Este Hilo es el encargado de verificar y mantener el estado de los circuitos según ha sido configurado en la ventana del horario de manera automática. Este hilo de programa es capaz de discernir los circuitos que se han activado de manera manual desde la aplicación móvil o desde la interfaz de escritorio. Ante un evento de estos (manual), la influencia de la ventana de activación o desactivación automática se desactiva por 24 horas y el circuito entra en un modo manual por 24 horas. .

#### 5.4.2. Remote Mode Handler

Este hilo de programa está encargado de verificar el estado de la conexión con el servidor para cambiar el modo de almacenamiento de la información, si es necesario. Si el dispositivo se desconecta, el sistema de almacenamiento cambia a la memoria local y si vuelve la conexión el handler es capaz de conectarse automáticamente al broker web para reportar los datos nuevamente, sin perdida de información.

#### 5.4.3. Mail Notification Handler

Este hilo es el encargado de verificar la fecha y realizar las notificaciones correspondientes al consumo eléctrico, el consumo de agua y el indicador de sostenibilidad. También, lleva el registro del consumo generado por fuera de los horarios recomendados por el Solar Decathlon.

#### 5.4.4. Data Report Handler

Esta rutina es la encargada de reportar automáticamente los valores medidos y almacenados en las gráficas, esta rutina almacena datos con una periodicidad que va desde cada segundo hasta cada 5 minutos. Como aclaración, cada vez que se reportan los datos de las gráficas, no se toman en cuenta todos los datos intermedios, por ende hay una perdida para variables de medición con una granularidad mayor a 1 segundo. En la figura 20 se puede observar que el periodo “T1” corresponde al tiempo de muestreo de alta granularidad a partir de la cual se grafican los datos en tiempo real en la ventana de medición; por otro lado, el periodo “T2” corresponde al tiempo de muestreo a partir del cual se guardan los datos en el servidor o en memoria.

#### 5.4.5. Indicator Handler

Esta es la rutina que se encarga de monitorear el estado de consumo eléctrico y de agua para modificar los valores de los indicadores de la ventana de inicio. Así como también, calcular el valor del indicador de sostenibilidad utilizando exclusivamente en el consumo eléctrico y de agua.

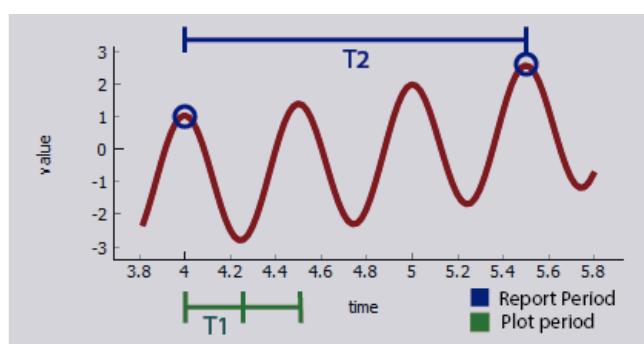


Figura 20: Descripción de los tiempos de muestreo utilizados en el sistema de adquisición. Fuente: propia

## 6. Serverless backend

Durante el desarrollo del *backend* se evaluó la posibilidad de utilizar las alternativas de nube más reconocidas y con mayor facturación en el mercado actual; Lo anterior con el objetivo de implementar el desarrollo con el proveedor más confiable. Dentro de los posibles proveedores se evaluaron las siguientes opciones: Amazon Web Services, Google Cloud y Microsoft Azure. Los anteriores proveedores fueron escogidos por el tamaño que tienen en la industria; una mirada de la repartición del mercado actual se puede observar en la figura ??

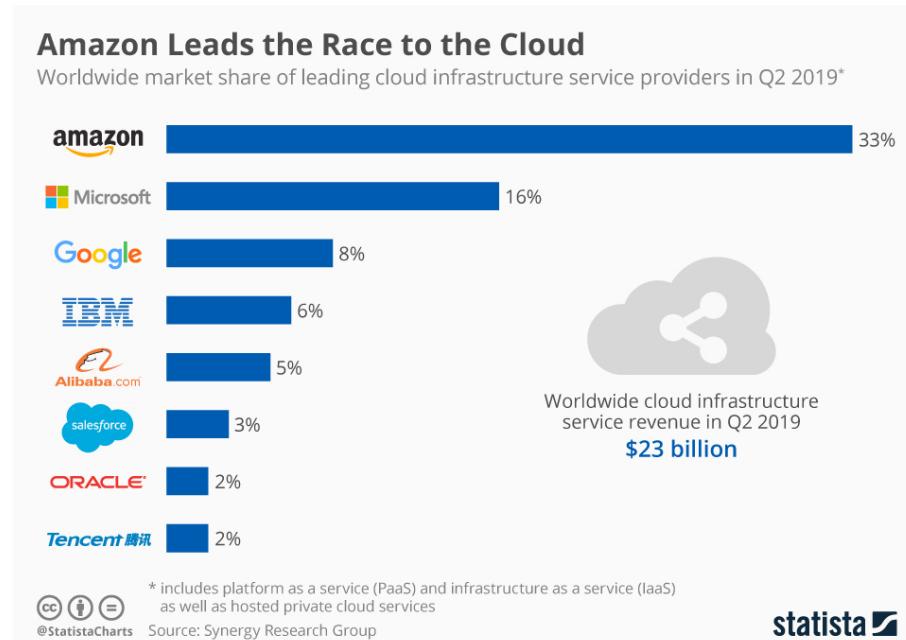


Figura 21: Estadística de la repartición del mercado de servicios de nube en 2019.  
Fuente: [2]

Teniendo en cuenta los tres proveedores mencionados anteriormente se pudo observar que los factores diferenciadores de cada uno, dependían estrictamente de los productos adicionales que estos ofrecían:

Por una parte, los servicios de nube de Amazon Web Services representan la mayoría del mercado de la nube, pero esto se debe en su mayoría al almacenamiento de información, por ende, tiene una gran variedad de servicios de nube, incluyendo sus tiendas en linea y sus servicios de storage webs. En resumen, Amazon posee la mayor variedad de servicios de nube en la actualidad.

Por otra parte, los servicios de Google incluyen el uso de su red privada que tiene el mejor desempeño en velocidad dentro del mercado. En adición, los servicios de

Google Cloud poseen interfaces de administración con más experiencia de usuario y la posibilidad de integrar servicios de machine learning, big data y de procesamiento de audio.

Finalmente, Microsoft Azure es el servicio de nube con la posibilidad de integrarse de manera optima con los servicios como Windows Server, Office, .Net, entre otros. Por otra parte la documentacion entorno a los servicios de microsoft Azure no es tan completas como las de los dos competidores anteriormente mencionados.

Cabe aclarar que al escoger a cualquiera de estos proveedores no resulta comprometido el uso de los servicios de otros: para integrar servicios de otro proveedor, solo se requiere establecer las condiciones apropiadas para autenticar nodos de red diferentes a los de las redes internas que los proveedores ya administran.

Teniendo en cuenta todo lo anterior, se escogió el servicio de nube de Google porque tiene los servicios de machine learning y de procesamiento de voz más avanzados. Por ende, aunque no se utilizan para esta version del proyecto, la integracion en futuros trabajos será más fluida. Además, debido a que el sistema operativo utilizado para probar la aplicacion celular es Android, la integracion de Google Cloud y la “User App” será mas fácil.

Una vez escogida la plataforma de nube y el servicio de comunicación se decidió desarrollar el *backend* tipo serverless para aprovechar todas las ventajas del servicio escalable de mensajes de internet de las cosas de Google llamado “Cloud Iot”.

### 6.1. Especificaciones técnicas

Para el montaje del modelo de la nube se utilizaron principalmente 4 componentes: la base de datos MySql, el servicio de almacenamiento por contenedores tipo “Storage 3” de Amazon, el servicio de Cloud IOT como bróker de internet de las cosas y las “Cloud Functions” del servicio de servidores de Google. De los elementos anteriormente mencionados se puede entender el backend como una serie de funciones que son activadas con eventos tipo solicitudes Https y mensajes basados en Mqtts, por ende todo el flujo de información será orquestado por las funciones que se definan en el sistema. El flujo de la información se puede observar más adelante en la sección “funciones de nube”. Adicionalmente, Un diagrama generalizado de los componentes del *backend*, se puede ver en el capitulo de diseño en la figura 2.

Las funciones de nube de Google operan como scripts de lenguajes de programación como Node.js, Python y Golang; estas requieren de librerías para operar de manera efectiva con cualquier componente del sistema. Para el desarrollo de este modelo se decidió utilizar el entorno de desarrollo de Node.js, puesto que era un lenguaje de

programación familiar y de alto nivel. Teniendo en cuenta lo anterior, para poder hacer uso de los componentes necesarios para implementar el sistema diseñado se utilizaron las siguientes librerías:

- “**@Google-cloud/storage**”: Esta librería es desarrollada y mantenida por google y permite hacer uso de las descargas, las subidas y la administración de los servicios de almacenamiento de archivos “Cloud Storage”.
- “**Mysql**”: La librería oficial de MySql para Node.js, con esta librería se implementó la comunicación con la base de datos MySql y se crearon todas las rutinas para guardar y obtener datos en la base de datos.
- “**Googleapis**”: Esta librería es la encargada de comunicarse directamente con cualquier dispositivo conectado a la red, fue utilizada para reenviar los mensajes recibidos por los dispositivos a través del puente “Cloud IoT”.
- “**Jsonwebtoken**” Esta librería fue utilizada para autenticar los mensajes recibidos por las solicitudes Https y Mqtts, a partir de las llaves públicas de los dispositivos registrados en el sistema. El algoritmo de encriptación utilizado fue el RSA256.
- “**Google-auth-lib**” La librería de autenticación de google fue utilizada para obtener las credenciales necesarias para utilizar el puente de IoT del proyecto.

## 6.2. Base de datos relacional y no relacional

Una parte muy importante del *backend* es la base de datos y, aunque el sistema de almacenamiento de “Cloud Storage” funciona como un disco duro con todas las facilidades para manejar archivos naturales, tiene limitaciones y costos asociados al número de operaciones de lectura y escritura que se pueden hacer en el mes. Por el contrario, la base de datos MySql tiene costos asociados únicamente con el tamaño del almacenamiento y la capacidad del procesador; debido a esto es la opción más apropiada para almacenar datos de pequeño tamaño pero de gran cantidad.

El diseño de la base de datos estuvo apoyado por el programa de administración de bases de datos: “*MySQL Workbench*”. Un esquema generalizado de la base de datos se puede observar en el diagrama 22, donde cada cuadro representa un grupo de tablas bajo el paradigma SQL.

Teniendo en cuenta lo anterior, se explicará por separado cada grupo de tablas con el objetivo de mostrar las escalabilidad del sistema y como fue implementada la base de datos pensando en los requerimientos funcionales de los productos House

## 6 Serverless backend

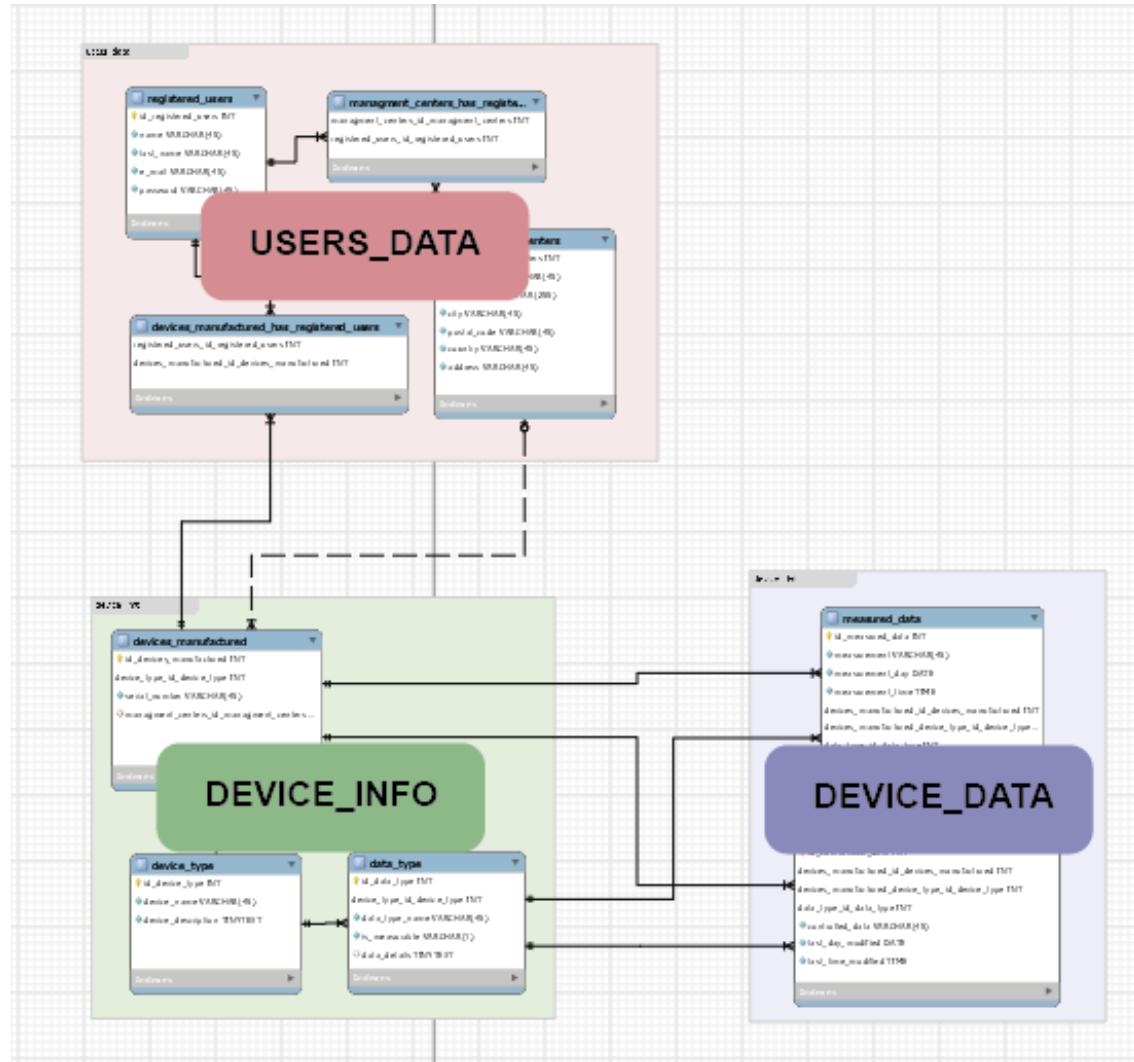


Figura 22: Diagrama general de la base de datos relacional MySql. Fuente: propia

Manger y User App. cabe aclarar que para representar los tipos de campos se utilizaron los siguientes símbolos: un bombillo amarillo significa que se trata de una columna de llave primaria única, un rombo azul representa un buffer de tipo *char* obligatorio, un rombo blanco significa buffer de tipo *char* no obligatorio y los campos sin símbolo son llaves secundarias de relación obligatorias.

- **Users data:** Figura 23. En este grupo de tablas se encuentran principalmente 2 tablas:

La tabla registered\_users, que almacena la información de los clientes; esta información consta de un correo de registro y un espacio para la “contraseña”. Realmente el espacio de la contraseña es una entrada para una “llave privada” generada por Google que se utiliza para acceder a la información personal de los usuarios desde el *backend*.

La tabla management\_centers, que agrupa los dispositivos fabricados en regiones de operación; en esta tabla se tiene como objetivo referenciar a los usuarios y los dispositivos a una zona horaria GTM y una descripción de dirección. Esta tabla presenta una utilidad representativa en el caso de realizar envíos o referenciar espacios geográficos en los cuales se encuentran los dispositivos.

Finalmente, Las tablas adicionales son las tablas de relación muchos a muchos de un paradigma SQL relacional.

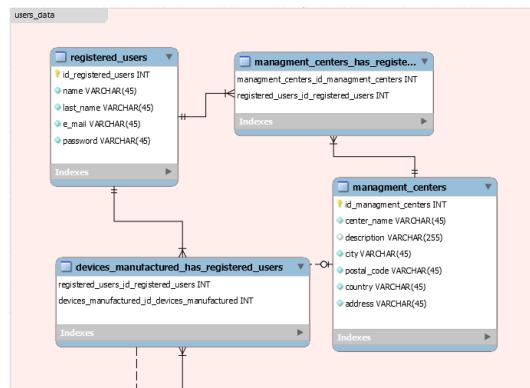


Figura 23: Grupo de tablas: users data. Fuente: propia

- **Device\_info:** Figura 24. En este grupo existen 3 tablas y todas contienen información relevante: La tabla devices\_manufatured contiene principalmente el número serial de un dispositivo para identificarlo a los ojos del fabricante. La tabla device\_type contiene una breve descripción del dispositivo y su nombre. Por último, la tabla data\_type sirve para almacenar la información de todas las

posibles variables medibles y los actuadores configurables para todos los dispositivos, este ultimo campo es para garantizar que el sistema pueda explicarse por si mismo con una simple consulta. Teniendo en cuenta que el sistema puede recibir dispositivos que no han sido fabricados de manera propia, juntando el nombre y el numero serial, se obtiene una identificación única por sensor o actuador conectado al sistema.

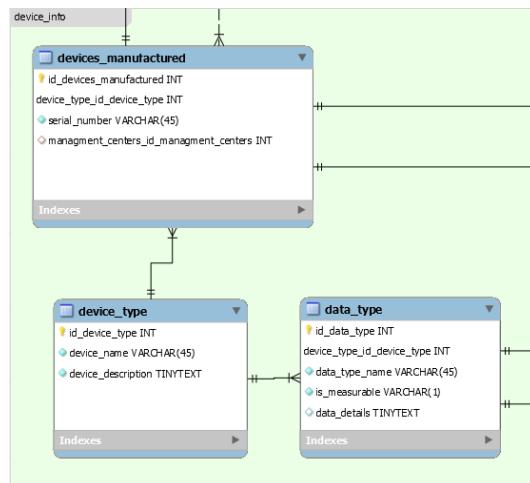


Figura 24: Grupo de tablas: device info

- **Device\_data:** Figura 25. El grupo de tablas de información se compone únicamente de 2; la tabla measured\_data contiene la información que fue medida y tiene una estampa de tiempo de mínimo 1 segundo, la tabla controllable\_data contiene las estampas de tiempo en las cuales se le dio la orden al dispositivo modificar su actuador de forma específica. Teniendo en cuenta lo anterior, un valor de control en un momento específico no garantiza que ese actuador se haya modificado de esa forma en el centro de gestión; garantiza que el dispositivo estaba conectado a la red y recibió el comando de hacerlo.

Finalmente el componente no relacional del sistema de almacenamiento tipo “Storage 3” funciona como un sistema de archivos y tiene como interfaz la api propia de google anteriormente mostrada. Internamente para almacenar la información de los usuarios se organizó la información en carpetas con el nombre “sql\_uidid mysql” donde el texto “id mysql” corresponde a la llave primaria única entera utilizada para identificar a los usuarios en la base de datos relacional (la columna llamada “id\_registered\_users” de la tabla “registered\_users”)

Teniendo en cuenta la estructura de almacenamiento anteriormente mencionada, el sistema obtiene su cuello de botella en el número de conexiones MySql que puede recibir, todos los demás elementos pueden escalar, en teoría, de manera infinita.

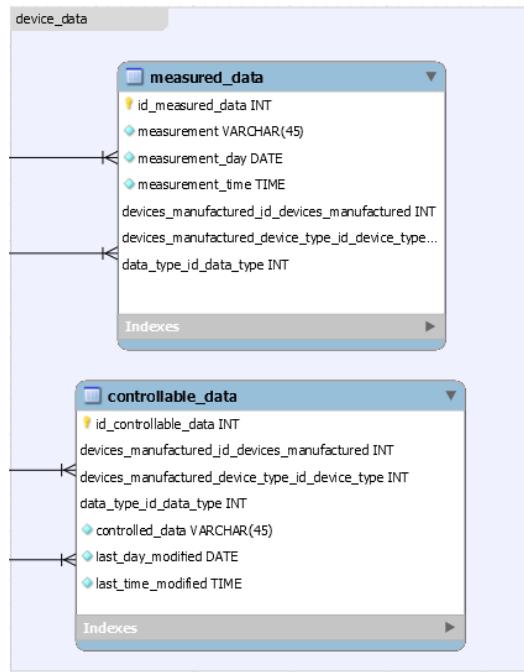


Figura 25: Grupo de tablas: device data. Fuente: propia

Para la versión de la base de datos que se está utilizando se pueden permitir 4000 conexiones simultáneas, por ende, se dice que el sistema está diseñado para 4000 dispositivos.

### 6.3. Relación de requerimientos funcionales

Debido a que este producto no se contempló en la establa de conceptualización y planeación del proyecto no tuvo una lista de requerimientos funcionales sobre los cuales crear scripts o funciones, por el contrario, los requerimientos funcionales de los otros dos productos: El House Manager y el User Manager exigían un desarrollo desde la perspectiva del *backend*. Teniendo en cuenta lo anterior, se explicaran los desarrollos realizados en el *backend* y los requerimientos funcionales que pretendían solucionar.

1. “*El usuario debe poder acceder a la información medida en tiempo real*”. Para cumplir este objetivo fue necesario crear un puente de conexión en tiempo real de un dispositivo a otro, siendo el segundo un dispositivo de monitoreo como lo es la User App. El puente es realizado por el “cloud iot” y a travez de una función de nube que es activada cada vez que se reporta un valor de medición (esta función será explicada más adelante)
2. “*El usuario debe poder acceder al histórico del mes y la relación de sus gas*-

*tos con ellos". Como solución a este requerimiento se diseñó una REST API (aquel conjunto de funciones basadas en solicitudes Http, que se utilizan para ofrecer un servicio en específico) capaz de obtener datos de la base de datos de medición o control entre un periodo de tiempo específico.*

## 6.4. Funciones de nube

Una función de nube es una rutina de código que se ejecuta cada vez que se realiza una solicitud Https o se recibe un mensaje directamente desde Google (como es en el caso del broker de IOT; ver figura 2 para ver bloques funcionales del *backend*) y esta función puede retornar un mensaje en formato de texto si así lo desea. Teniendo en cuenta lo anterior, las funciones de nube implementadas en este sistema tienen como características: una memoria ram de 512 megabytes, la posibilidad de tener acceso a una memoria volátil temporal únicamente existente durante el tiempo de ejecución de la función y la posibilidad de ejecutar los lenguajes de programación: Node.js, Python y Go. Cabe aclarar que este tipo de arquitectura posee como limitación un límite de ejecución por función de 9 minutos. Las anteriores características fueron configuradas directamente en la plataforma de Google y se encuentran dentro de los valores gratuitos del servicio.

Teniendo en cuenta lo anterior, se implementaron 4 funciones de nube que le permitieron al sistema cumplir con los requerimientos funcionales. Estas cada una de las funciones serán explicadas en los siguientes apartados.

### 6.4.1. Get\_historical\_data:

Esta función fue diseñada para recibir un mensaje a través de un “post request”. El contenido del post debe ser buffer codificado a 64 bits con un objeto de JavaScript (Json) convertido. El contenido de este objeto debía ser el necesario para: validar la legitimidad de la solicitud, conocer el tipo de dato que se desea obtener de la base de datos, el dispositivo del cual se desea conocer la información y el rango de tiempo en el que se desean consultar los datos. Teniendo en cuenta lo anterior un ejemplo del objeto Json aceptado como argumento de la función es:

```
{  
    "type": "control",  
    "sql_uid": 1,  
    "number_points": 10,  
    "device_name": "House Manager SDL"  
    "serial_number": 1000023211  
    "data_type": "potencia activa instantanea",  
    "init_date": "2019-07-18",  
    "final_date": "2019-08-12",  
}
```

```
        "encrypted_token": "SsidfmjEFShfDBGSsdEFSBzdfw3RQEfassdg23"
    }
```

Donde el campo “type” es opcional y tiene como valor por defecto los datos de medición, el campo “sql\_uid” corresponde al identificador único de usuario de la base de datos, el campo “number\_points” corresponde al número máximo de puntos que puede tener el objeto de respuesta, “serial\_number” es el numero serial del dispositivo que genero los datos que se desean obtener, “data\_type” es un string con el nombre del tipo de dato registrado en la tabla data\_type, los campos “init\_date” y “final\_date” representan los límites temporales en los cuales se debe realizar la consulta.

Por último, el campo “encrypted\_token” se debe incluir un String encriptado conteniendo el objeto de autenticación. El String debe estar encriptado utilizando el algoritmo de Rivest–Shamir–Adleman de 256 bits y el objeto descifrado debe contener la siguiente información:

```
{
    iat: 1234232009,
    exp: 1236323009
}
```

Donde “iat” es un entero con el número de milisegundos en los cuales se creó el token, teniendo en cuenta, un punto de referencia de la maquina (Enero 1 de 1970 a la hora 00:00:00 UTC.) y el campo “exp” es también un entero con los milisegundos en los cuales expira el token, de autenticación, teniendo en cuenta el mismo punto de referencia.

Debido a que el sistema de autenticación basado en los JWT es el mismo en todas las funciones no será explicado nuevamente en las siguientes secciones, solo serán explicados los campos nuevos o los que tengan un significado diferente.

El algoritmo de la función se puede separar en 5 pasos generales, fig 26, cada uno de los pasos se puede ver a continuación:

1. **Segmentacion del tipo de request:** En esta etapa de la función se verifica que la solicitud sea únicamente de tipo POST
2. **Segmentacion por formato de mensaje:** En esta etapa se verifica que el tipo de contenido del mensaje sea “application/json”
3. **Adquisicion de la llave publica:** Para autenticar el usuario se accede al servicio de almacenamiento S3 para optener la llave “verificadora” del usuario en cuestión.

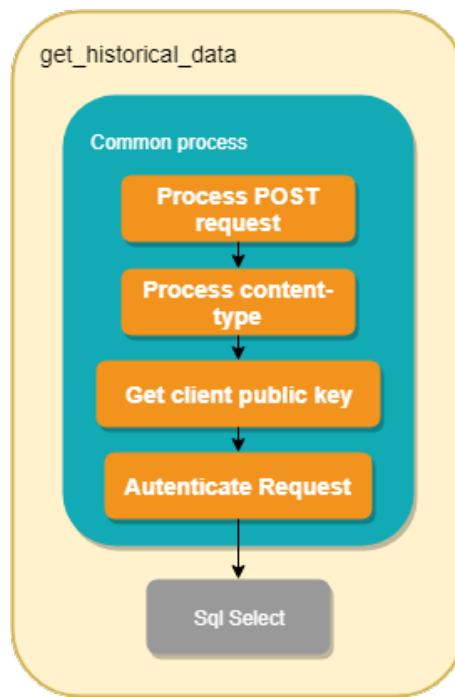


Figura 26: Algoritmo de la función: Get historical data. Fuente: propia

4. **Autenticación de usuario:** Se descifra el token de acceso y se verifica que el token no se encuentre vencido.
5. **Consulta SQL:** Se utiliza la información contenida en el mensaje para realizar una consulta SQL dependiendo del tipo de operación y se organizan los datos en un objeto resultado.

Teniendo en cuenta el proceso anterior, un ejemplo del objeto resultante de solicitud Http a esta función es:

```
{  
    ``data\_\_type``: ``potencia activa instantanea,  
    "data": [  
        {  
            "data": "1213123",  
            "date": "2019-06-12",  
            "time": "17:44:51",  
        },  
        {  
            "data": "333123",  
            "date": "2019-07-12",  
            "time": "3:12:11",  
        }  
    ]  
}
```

```
        }
    ]
}
```

Donde el objeto es un Json codificados en base 64 del objeto convertido a String. el contenido de "data" depende del número de puntos que se soliciten y el campo "data\_type". Teniendo en cuenta el vector de datos de ejemplo, cabe aclarar que toda la información temporal recibida esta referenciada a la zona horaria cero (UTC: 0).

#### 6.4.2. Send\_command:

Esta función también fue diseñada para recibir un mensaje a través de un "post request". y también debe recibir un buffer codificado a 64 bits con un objeto de JavaScript (Json) convertido. La razón por la cual se escogió el "POST" como el tipo de solicitud fue puesto que con las solicitudes post el servidor en cuestión encripta la información usando su certificado de seguridad SSL, es decir, que en este caso la información transmitida esta encriptada con el certificado SSL de Google.

Teniendo en cuenta que la función también fue diseñada basada en un post y con todos los requerimientos de encriptación y codificación, un ejemplo de un mensaje seria:

```
{
  "sql_uid": 1,
  "mqtt_sub_folder": "control/raspberry",
  "operation": "send-save",
  "encrypted_token": "SsidfmjEFShfDBGSsdEFSBzdfw3RQEfassdg23",
  "message": {
    "some_json_object"
  }
}
```

Dado el anterior objeto Json: el campo de "mqtt\_sub\_folder" define el tema al cual el dispositivo de recepción está suscrito. El campo "operation" contiene una de las siguientes opciones; send-save, send o save y controla la operación que debe realizar el *backend* con el mensaje. En caso de ser save, solo guardará en la base de datos el mensaje enviado (este mensaje debe contener la información necesaria para guardarse en la base de datos; si no tiene el formato indicado, no será guardado en la base de datos a pesar de que si sea enviado y recibido por el dispositivo). En caso de solo ser send, el *backend* le enviará un mensaje al dispositivo, si es posible, y finalizará la operación. Finalmente, si la operación es send-save el *backend* intentará enviarle un mensaje al dispositivo, si lo recibe, asume que la variable de control fue modificada y guarda en mensaje en la base de datos.

El campo message contiene el mensaje que recibirá el dispositivo, teniendo en cuenta que el mensaje es para la modificación de una variable de control el mensaje debe contener la estampa de tiempo en la que se generó la orden, el valor al cual se desea modificar la variable y el tipo de data a la cual corresponde la información. Un ejemplo de Json con la capacidad para almacenar la información de la orden es:

```
"some_json_object" = {  
    "device_name": device_name,  
    "serial_number": serial_number,  
    "data_type": ["control circuito a", "control circuito b"],  
    "data": ["off", "on"],  
    "date": ["2019-06-12", "2019-07-12"],  
    "time": ["3:12:11", "3:12:40"],  
}
```

Como se puede observar anteriormente, los campos de "data", "datez" "timecontienen un vector, lo que le permite a esta función realizar múltiples operaciones SQL y modificar múltiples actuadores con una sola solicitud.

El resultado de esta operación es un objeto tipo Json con la información de la operación SQL, es decir, información como el número de filas afectadas, el tiempo que tomo la operación, entre otros. Para más información de este objeto se puede revisar la documentación oficial de la librería de mysql [9].

Adicionalmente, un diagrama generalizado del algoritmo se puede ver en la figura 27. Teniendo en cuenta esta figura el algoritmo de la función se puede dividir en 6 pasos:

1. **Segmentación del tipo de request:** Al igual que en la función anterior se discrimina para que solo se acepten solicitudes tipo "POST"
2. **Segmentación por formato de mensaje:** Así como en la función "get\_historical\_data", se rechazan las solicitudes que no tengan como formato de contenido "application/json".
3. **Adquisicion de la llave publica:** También, hacemos la consulta al servicio de almacenamiento para adquirir la llave pública que será usada en el proceso e descifrado.
4. **Autenticación de usuario:** Nuevamente Se descifra el token de acceso y se verifica que el token no se encuentre vencido.
5. **Envío del mensaje Mqtt:** Se envía exactamente el contenido del campo mensaje al dispositivo de interés usando el tópico especificado en el campo ("mqtt\_topic")

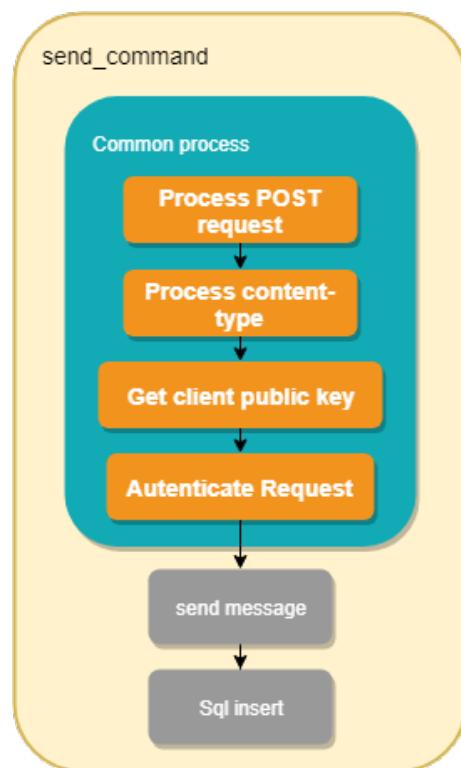


Figura 27: Algoritmo de la función: Send command. Fuente: propia

6. **Registro de la información:** si el dispositivo está conectado y ha recibido correctamente el mensaje, se guarda la información del mensaje en la base de datos, si el mensaje no tiene el formato indicado este último paso no se realiza.

#### 6.4.3. IOT\_gateway\_control:

Esta función es muy diferente a las otras dos funciones explicadas anteriormente; por el contrario a ellas, esta función es activada o ejecutada a través de un sistema de publicación y suscripción interno de Google Cloud (“pub/sub message system”) para comunicar sus diferentes dependencias. En el momento que un bróker Mqtt recibe una publicación bajo el tema “control” un evento de pub/sub se genera, y por consiguiente, esta función de nube que está asociada a él.

Adicionalmente, el proceso y funcionamiento de la función puede considerarse igual al de las otras dos funciones con pequeños cambios, todo el proceso de autenticación es manejado directamente por el servicio de “Cloud IoT”, por ende el campo “encripted\_token” ya no se encuentra en los mensajes recibidos. Un ejemplo de un mensaje enviado por un dispositivo a través de una publicación Mqtt es:

```
{  
    "sql_uid": 1,  
    "mqtt_sub_folder": "control/raspberry",  
    "operation": "send-save",  
    "message": {  
        "device_name": "House Manager SDL",  
        "serial_number": 1000000000,  
        "data_type": ["circuito 1", "circuito 2"],  
        "data": ["on", "off"],  
        "date": ["2019-06-12", "2019-10-0"],  
        "time": ["17:44:51", "21:00:51"],  
    }  
}
```

Teniendo en cuenta el Json anterior se puede observar que su formato es muy parecido al del objeto de la función “send\_command” con la diferencia que no contiene el campo de objeto encriptado, esto se debe a que el objeto encriptado es utilizado como contraseña al momento de establecer la conexión con el bróker desde el cliente. En adición, un diagrama generalizado del algoritmo se puede ver en la figura 28. Teniendo en cuenta esta figura el algoritmo de la función se puede dividir en 4 pasos:

1. **Organizar el mensaje:** Los mensajes recibidos por el broker Mqtt de Google tienen un formato tipo “buffer”, por lo tanto, en esta etapa se convierte el arreglo recibido en un formato interpretable por el sistema.

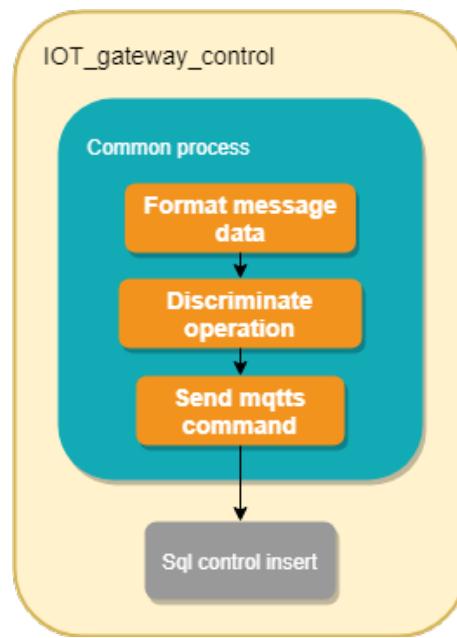


Figura 28: Algoritmo de la función: Iot gateway control. Fuente: propia

2. **Discriminación de operacion:** Así como se discriminó en la función “Send\_command” en este campo se describe el tipo de operacion que se debe realizar con el mensaje. La operacion será “send-save” por defecto
3. **Envío del mensaje de control Mqtt:** Se envia exactamente el contenido del campo mensaje al dispositivo de interés a travez de su tema (“mqtt topic”).
4. **Registro de la información:** si el dispositivo está conectado y ha recibido correctamente el mensaje, se guarda la información del mensaje en la base de datos, si el mensaje no tiene el formato indicado este último paso no se realiza.

#### 6.4.4. IOT\_gateway\_measure:

Esta función comparte todos sus pasos con la anterior, es decir que es muy parecida desde una perspectiva general; la diferencia se encuentra en el último paso de la misma, un diagrama generalizado de este algoritmo se puede ver en la figura 29. En el último paso, si todos los anteriores fueron completados exitosamente, el algoritmo intenta guardar la información en la tabla de control de la base de datos.

Adicionalmente, la función debe recibir un Json con el mismo formato que el recibido por “iot\_gateway\_control” con diferencias en sus campos: “data\_type” y “mqtt\_sub\_folder”, puesto que en el tipo de dato debe contener el identificador de la variable de control y en mqtt\_sub\_folder debe haber un String que identifique el tópico Mqtt que el dispositivo destino pueda reconocer como un mensaje de control.

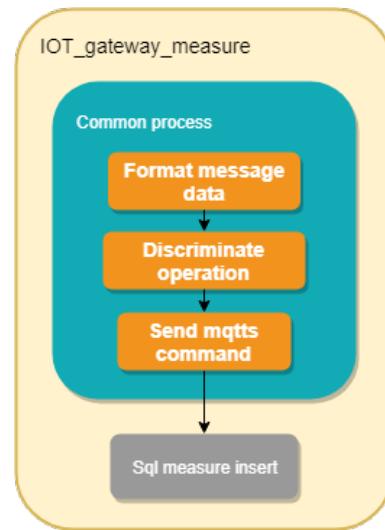


Figura 29: Algoritmo de la función: Iot gateway measure. Fuente: propia

Para efectos prácticos, en esta etapa de la implementación, con el objetivo de discernir un tipo de mensaje de otro se escogió como formato de tema: “nombre/control”, donde nombre es el nombre del dispositivo al cual se le desea enviar el mensaje. Con esto, no se quiere decir que el tema Mqtt esté ligado a este formato, en el caso que se deseen escalar los comandos y añadir más actuadores a un dispositivo ya diseñado, se puede modificar el sistema de temáticas propuesto para esta versión de los productos *backend*.

## 7. User app

Para el desarrollo de este sistema se decidió implementar la aplicación utilizando nuevas tecnologías de desarrollo multiplataforma para móviles como: React native, Flutter, Ionic. Lo anterior debido a que, dado que se desea proyectar la aplicación a una escala mundial, es necesario desarrollar aplicativos para ambos sistemas operativos (Ios y Android), pero tener un equipo de desarrollo de ambas plataformas resulta mas costoso que tener un equipo de una tecnología híbrida.

Teniendo en cuenta lo anterior, los frameworks para el desarrollo de aplicaciones móviles híbridas más famosos en la actualidad son React Native, Flutter e Ionic.

Por una parte, React Native es un framework open source de Javascript creado y mantenido por Facebook Inc y posee 78,400 estrellas en el repositorio publico en Github [10].

Por otro lado, Flutter es un framework para el lenguaje de programación Dart. Ambos Dart y Flutter son creados y mantenidos por Google y el repositorio de Flutter posee 68,000 estrellas en Github[10].

Finalmente, Ionic es un framework de Javascript creado y mantenido por la corporacion Drift Co. Su repositorio en Guithub tiene 39,100 estrellas.

Teniendo en cuenta lo anterior, se escogió el framework con mayor acogida por la comunidad de desarrolladores, es decir, React Native. En adición, la desición tambien tuvo en mente la necesidad alejarse de productos no estables, puesto que los frameworks de desarrollo móvil multi plataforma son muy nuevos y pueden existir bugs no reportados. Por ejemplo, Flutter, no tiene una versión estable soportada por Google.

React Native permite desarrollar aplicaciones nativas para Android y iOS a través de un sistema de traducción de componentes desde JavaScript al lenguaje nativo de la maquina (Java para Android y Swift para IOS). Por ende, las aplicaciones desarrolladas en este framework tienen un desempeño muy similar a las implementadas directamente en los lenguajes nativos correspondientes.

Además, una ventaja estratégica que tiene el desarrollar aplicaciones a través de este framework es que ambos desarrollos (para IOS y Android) quedan con las mismas interfaces gráficas, lo que le da al cliente la misma experiencia de usuario sin importar en que plataforma se encuentre.

Aunque desarrollar aplicaciones para Android y IOS con el mismo lenguaje de pro-

gramación suena extremadamente ventajoso, hay limitaciones técnicas que serán mencionadas más adelante en este capítulo.

## 7.1. Especificaciones técnicas

Para poder realizar el aplicativo se utilizaron diferentes librerías de JavaScript creadas específicamente para el framework de React-Native. Las librerías utilizadas en el aplicativo fueron:

- **React-native-google-signin:** Esta librería fue utilizada para obtener las llaves privadas y la información de usuario utilizando el servicio de autenticación de Google.
- **React-native-progress:** Con esta librería se lograron desarrollar los indicadores de progreso circulares para mostrar el índice de consumo eléctrico de agua y el coeficiente de sostenibilidad.
- **Jsrsasign:** Usando esta librería fue posible encriptar los JWT de autenticación a partir del protocolo de RSA de 256 bits. Esta librería fue de vital importancia puesto que poseía toda la compatibilidad necesaria para ser ejecutada con JavaScript puro, sin el uso de paquetes adicionales escritos en Java o Swift.
- **React-native-svg:** Esta librería fue utilizada para apoyar el desarrollo de componentes visuales adicionados a las gráficas.
- **React-native-svg-charts:** Usando esta librería fue posible integrar gráficas en el aplicativo móvil.
- **React-native-swiper:** Con esta librería se implementó la interfaz tipo deslizante (“swipe”) en las ventanas de introducción de la aplicación.
- **React-native-vector-icons:** Con esta librería fue posible utilizar los iconos nativos de Android y IOS como el botón de menú, de home, entre otros.
- **Toggle-switch-react-native:** Con esta librería se implementaron botones conmutadores (on off), utilizados en la escena de control.
- **React-navigation:** Esta librería fue utilizada para establecer el control de la navegación y el menú dentro de la aplicación como se puede ver en la figura 30.
- **React-redux:** Usando esta librería fue posible comunicar de manera efectiva todos los objetos y componentes del sistema.
- **Fetch:** Con esta librería se estableció la comunicación basada en solicitudes Https con el backend.

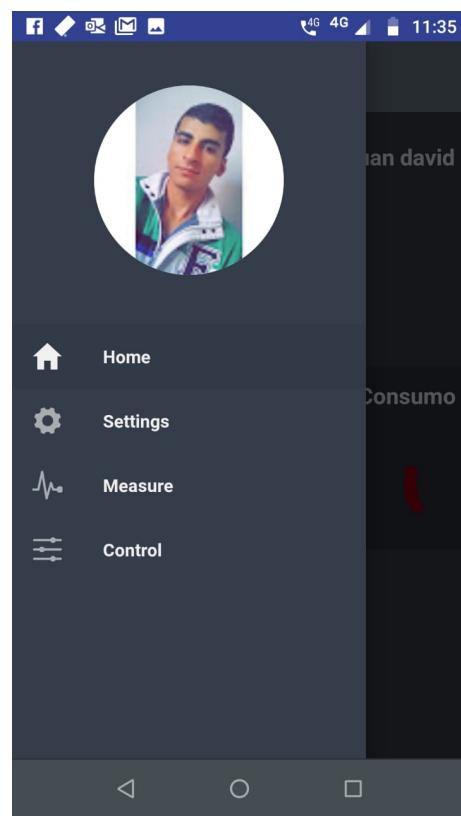


Figura 30: El menú desplegable con las aplicaciones actualmente desplegadas. Fuente: propia

Teniendo en cuenta que React\_native traduce todos sus componentes al lenguaje nativo del celular, es necesario entender que no todas las librerías de JavaScript se pueden usar para el framework, únicamente aquellas que tienen incluido el enlace de bajo nivel con ambos sistemas operativos. Por ende, las librerías con mayor compatibilidad son las que están desarrolladas exclusivamente con JavaScript puro.

Con la anterior consideración en mente, en el proceso de desarrollo se presentó un problema por asumir incorrectamente que un “snipet” (Archivo o script que cumple una función en específico) era capaz de funcionar en el entorno de programación de React-Native. Este pequeño “snipet” fue desarrollado pensando en integrar las funcionalidades del protocolo de comunicación Mqtts a la aplicación; pero usaba librerías que estaban basadas en paquetes de bajo nivel equivocados, es decir, para computadores de uso comercial como Linux o Windows. Por ende, las funcionalidades de tiempo real fueron implementadas utilizando únicamente el protocolo de comunicación Https.

Para finalizar, de las librerías utilizadas, se puede observar que la mayoría son para mejorar la interfaz gráfica, mejorar la experiencia de usuario u obtener la misma vista entre sistemas operativos Android y Ios.

## 7.2. Requerimientos Funcionales

Debido a que durante el proceso de planeación del proyecto también se utilizó la tabla de requerimientos 1 para definir las especificaciones de esta aplicación, y estos requerimientos guiaron el proceso de desarrollo, se explicará cómo se implementaron las especificaciones establecidas:

1. *“El usuario debe poder acceder a la información medida en tiempo real.”* Para solucionar este requerimiento, en una primera etapa de desarrollo, se optó por utilizar una librería llamada “react-native-mqtt”. Esta librería funcionaría como puente entre las librerías: “Paho Mqtt Client” de Android y “Mqtt Framework” de IOS, pero debido al estado tan pionero de las tecnologías de desarrollo de React Native y el protocolo de comunicación Mqtt, la librería no era estable y muchas otras similares tampoco.

Teniendo en cuenta lo anterior se implementó la comunicación en ambos sentidos a través de las solicitudes Https, por ende, para obtener los datos en tiempo real en la aplicación le realiza solicitudes al backend de tal forma que reciba los últimos datos guardados en el sistema de almacenamiento MySql.

Finalmente, para mostrar la información medida en tiempo real se implementó la escena de medición “measure” tal como se ve en la figura 31, donde des-

lizándose hacia abajo se puede tener acceso a las gráficas con contenido de otros sensores.



Figura 31: Escena para la medición en tiempo real de las variables. Fuente: propia

2. “*El usuario debe ser capaz de cambiar los parámetros de configuraciones establecidos por defecto para la vivienda del solar Decathlon:*” Por defecto los parámetros de la vivienda del Solar Decathlon involucran unos horarios de uso de las cargas eléctricas, por ende, cuando el usuario realiza un cambio en la aplicación celular el sistema en sitio desactiva su configuración predeterminado.

Finalmente, se implementó una escena de configuración. Esta escena, no se añadió ninguna entrada de usuario y se dejó para ilustrar la posibilidad de incluirlo en el futuro. La escena se puede observar en la figura 32. En conclusión, no se cumplió este requerimiento.

3. “*El usuario debe poder acceder al histórico del mes y la relación de sus gastos con ellos:*” Para implementar este requerimiento se utilizaron barras circulares de progreso que indican la cantidad de energía o agua que ha consumido el usuario en el día, comparándola con un valor promedio. La escena desarrollada para solucionar este requerimiento se puede ver en la figura 36 donde el usuario

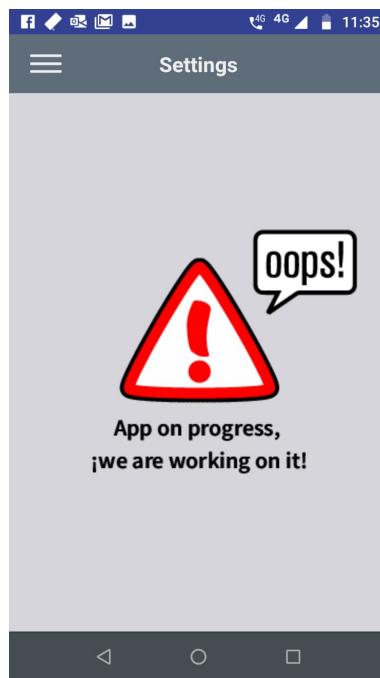


Figura 32: Escena de configuraciones, actualmente no disponible. Fuente: propia

puede deslizar horizontalmente el centro de la pantalla para revelar los otros indicadores.

4. “*La aplicación notificará al usuario cuando la factura este vencida:*” para este requerimiento se implementó una función adicional en el aplicativo en sitio, por ende, el requerimiento se cumplió desde la aplicación de escritorio.
5. “*El usuario debe ser capaz de poder ver su impacto ambiental basado en datos cualitativos y cuantitativos:*” Teniendo en cuenta la figura 33 uno de los indicadores corresponde a un indicador de sostenibilidad que está basado en la huella de carbono del consumo de agua y el consumo eléctrico, pero el componente cualitativo fue incluido en la notificación e-mail al fin del mes.
6. “*El sistema debe ser capaz de notificar las perdidas eléctricas y por consiguiente económicas de un mal uso de los horarios establecidos por defecto:*” Para implementar este requerimiento se desarrolló en el aplicativo en sitio la función de notificación automática. Es decir las notificaciones por perdidas no fueron incluidas en esta aplicación.

### 7.3. Funciones adicionales

En adición a los requerimientos funcionales se añadieron características a la conceptualización de diseño original, lo anterior, para mejorar el funcionamiento de la

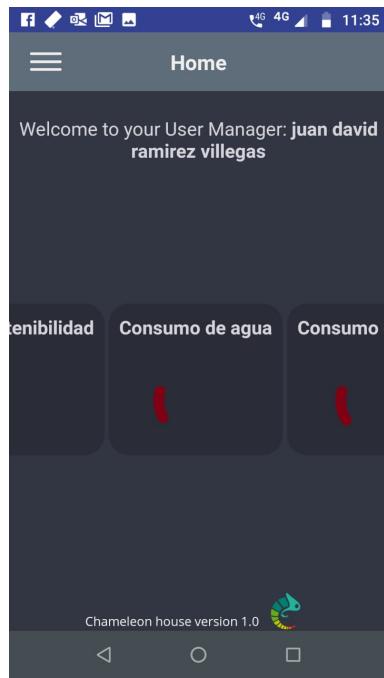


Figura 33: Escena para la visualización del consumo histórico. Fuente: propia

plataforma y la experiencia de usuario. Estas funciones se pueden ver a continuación:

1. **Pantalla de control:** Teniendo en cuenta un diseño con el mayor control y administración de los actuadores y sensores conectados al sistema, para esta versión de la aplicación, se implementó una interfaz de control que se puede observar en la figura 34, donde el usuario tiene la posibilidad de modificar la configuración horaria programada por defecto para el Solar Decathlon. Una vez el usuario altera el estado de los circuitos de la vivienda a través de la ventana, El sistema de agenda local se desactiva por 24 horas para darle la prioridad a lo que defina el usuario.
2. **Ventana de introducción:** Esta función es más un componente decorativo, la escena le da la bienvenida al usuario, mejora la experiencia de usuario y le da status a la interfaz. Figura 35.
3. **Servicio de autenticación:** Finalmente, se implementó un servicio de autenticación utilizando la api de Google, utilizando su api de autenticación se desarrolló la interfaz para obtener la información del usuario como: correo, numero de teléfono, nombre completo y una llave encriptada con la que la app puede re autenticar el usuario en caso de ser necesario tal como se ve en la figura 36.

## 7 User app

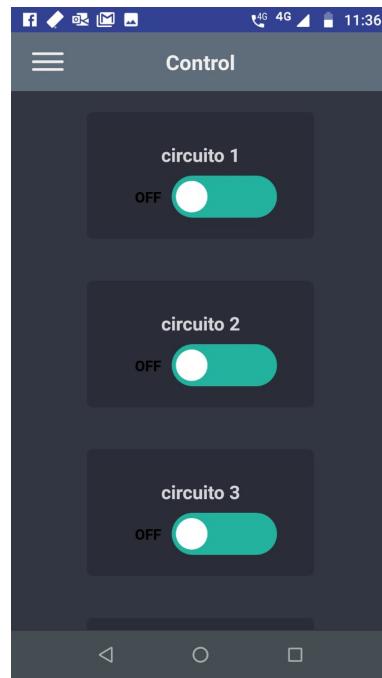


Figura 34: Escena para el control de los circuitos de la vivienda. Fuente: propia



Figura 35: Escena de bienvenida 1. Fuente: propia

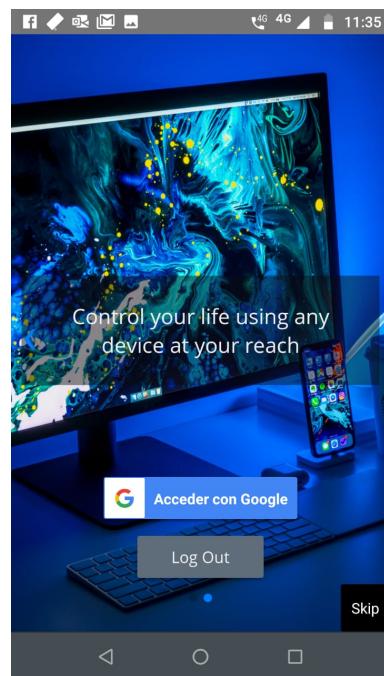


Figura 36: Escena de inicio de sesión. Fuente: propia

## 8. Pruebas y resultados

### 8.1. Pruebas unitarias

Las pruebas explicadas en este segmento involucraron clases internas de los 3 grandes programas del proyecto: el backend, la User App, y el House Manager. Es decir, se probaron los componentes más simples del sistema como se muestra a continuación:

- **House Manager:** Las pruebas unitarias explicadas a continuación involucraron paquetes de terceros y rutinas propias.
  - El documento “\_\_prueba spectrum.py” contiene la prueba de renderizado de los paquetes de Python Pyqtgraph y Pyside juntos en una misma ventana.
  - El archivo “smtp\_test\_privateemail.py” contiene la prueba para verificar el proceso de envío de e-mails utilizando el dominio “alfagenos.com”.
  - El documento “test\_http\_request.py” fue utilizado para realizar pruebas de comunicación a través de solicitudes Https tipo post desde el lenguaje de programación de Python.
  - El programa “test\_mqtt.py” prueba el componente de comunicación del House Manager que utiliza el protocolo de comunicación mqtt.
  - El archivo “test\_oauth.py” prueba el servicio de autenticación de Google y el manejo de las credenciales; su creación y eliminación.
- **User app:** Las pruebas unitarias de esta aplicación se realizaron renderizando objetos en una ventana de prueba desactivada para la versión de lanzamiento. Los objetos probados fueron los contenidos en los archivos: ControlScreen.js, MeasureScreen.js, IntroScreen2.js, Timer.js, MyChart.js
- **Backend:** Las pruebas del backend se realizaron a los diferentes microservicios implementados. Los scripts de prueba fueron:
  - El programa “test\_control\_SENDER\_SUBSCRIBER.js” se utilizó para probar el servicio de control del backend desde la perspectiva del generador y receptor de la orden.
  - El archivo “test\_measure\_SENDER\_SUBSCRIBER.js” contiene la prueba para verificar el proceso de reporte de datos.
  - El programa “test\_get\_history.js” fue utilizado para probar el servicio de solicitud de datos para cierto rango de tiempo.
  - El archivo “test\_send\_command.js” prueba el servicio de envío de mensajes mqtt a dispositivos específicos a través de solicitudes Https tipo “post”.

## 8.2. Pruebas de integración

Para finalizar, debido a que el sistema desarrollado está constituido enteramente por software se decidió adicionar un componente de hardware para probar conceptualmente la capacidad de añadir cualquier sistema de medición o actuación.

Como dispositivo de medición se utilizó un medidor bifásico de la empresa Inelca con una interfaz de comunicación Rs485. Utilizando esta interfaz y un conversor USB a Rs485 se logró comunicar la Raspberry con el medidor de Inelca.



Figura 37: Dispositivo Inelca utilizado para la prueba de concepto. Fuente: propia

Para finalizar, se puso a prueba el sistema añadiendo la función de adquisición necesaria para obtener el valor del voltaje de valor cuadrático medio (Rms) de la linea U1 del medidor. La función utilizaba un rutina a partir del cual se generaban y se interpretaban las tramas para adquirir el valor de voltaje del medidor. Una foto de todos los sistemas funcionando en conjunto se observa en la figura 38.



Figura 38: Prueba de concepto usando todos los sistemas. Fuente: propia

### 8.3. Resultados

A lo largo de este proyecto de grado, se construyó un sistema de administración de datos “serverless” para sensores y actuadores. Para ello se utilizaron los protocolos de comunicación Https y Mqtts y los servicios de almacenamiento SQL y Google Cloud Storage. La selección de estas herramientas le permitió al sistema monitorear y controlar los sensores y actuadores, conectados al sistema desde la Raspberry, de manera local y remota. La estructura en la cual se implementó el proyecto permitió que contara con las siguientes características:

- Los tiempos de respuesta del backend fueron en promedio de 1.31 segundos para el proceso mas demorado del sistema: reportar un dato, almacenarlo y enviarlo al dispositivo necesario usando Mqtts. Lo anterior teniendo en cuenta que el servidor de Google utilizado se encuentra en San Francisco USA.
- Las interfaces gráficas de ambos clientes (Escritorio y móvil) tienen el mismo estilo de diseño y son multiplataforma. Por una parte la aplicación de escritorio puede funcionar en Windows y Linux, por otra parte la aplicación móvil puede funcionar para Android y Ios.
- La estructura utilizada y la manera en la que se modeló el sistema hacen que éste sea escalable, es decir, posee la capacidad de expandirse en cuanto a número de dispositivos y usuarios. En teoría, el sistema puede soportar 4000 dispositivos conectados al mismo tiempo y una cantidad de usuarios limitada por el espacio de memoria comprado para los servicios SQL y Google Cloud Storage.
- El sistema fue diseñado teniendo en cuenta dos niveles de cifrado: el primer nivel consta de la encriptación manejada directamente por el proveedor de servicios (Google) y el segundo consta del cifrado de las credenciales de los dispositivos utilizando el método RSA-256. En adición, respecto a la seguridad de los datos personales del usuario como numero celular correo, etc, el servicio de Google se encarga directamente de hacer la encriptación y almacenamiento de los mismos.
- La selección del proveedor de servicios (Google) le permitió al sistema estar protegido de ataques de denegación de servicio de nivel 4 o inferior como: inundaciones SYN, inundaciones de fragmentos de IP, agotamiento de puertos, entre otros.

Teniendo en cuenta el proceso de implementación y los resultados obtenidos, el sistema cumplió en un 81 por ciento los requerimientos funcionales esperados. Los requerimientos que no fueron satisfechos fueron el 1.3 y 1.7. El primero no fue implementado directamente en la aplicación móvil pero, a pesar de que no era un requerimiento como tal, se implementó en la aplicación de escritorio. El segundo no fue implementado en ninguna de las dos aplicaciones.

Dentro de la carpeta del proyecto del “House Manager” se pueden encontrar scripts que fueron importantes para el despliegue de la aplicación. Por ejemplo; en el directorio “./src/installing/” se encuentran los ejecutables para la instalación de la aplicación en Windows y Debian; en el directorio “./src/test/” se encuentran las rutinas de “testing” utilizadas para algunos scripts y objetos utilizados en el programa.

Adicionalmente, en la carpeta del proyecto del *backend* se pueden encontrar los scripts de prueba de cada una de las funciones de nube. Por ejemplo; en el directorio “./src/CLOUD\_device\_control/” se encuentran la prueba unitaria con el nombre “test\_control\_SENDER\_SUBSCRIBER.js” de la función contenida en la carpeta. En cada una de las carpetas de cada función de nube se encuentra su respectiva prueba unitaria.

Finalmente, en la carpeta del proyecto del *User App* se puede encontrar en el directorio raíz las ventanas de prueba: “test\_App.js” y test\_App2.js que fueron utilizadas para probar los componentes ubicados en el directorio “./src/components/”. No todos los componentes de esta carpeta fueron utilizados en la aplicación final, puesto que no pasaban las de renderizado; esto se debe a que eran componentes basados en librerías abandonadas o dañadas.

## 9. Conclusiones y Trabajos futuros

### 9.1. Conclusiones

- Cuando se realizan aplicaciones de escritorio, como la desarrollada en este documento, con tecnologías como Java o Python existen pocas herramientas para mejorar la experiencia gráfica de la interfaz. Por el contrario, en un entorno web, la comunidad mejora el sistema de interfaces a una velocidad excepcional. Teniendo en cuenta lo anterior, desarrollar una aplicación de escritorio basada en alguno de estos dos lenguajes es viable si la aplicación es pequeña, sin embargo, si se desea escalar el aplicativo, es más apropiado utilizar un frameworks web con un backend local.
- Dentro de las posibles maneras de implementar productos, que necesiten algún tipo de arquitectura web como backend, el diseño basado en los servicios serverless de algún proveedor como; Amazon, Google o Azure, es la mejor alternativa si la empresa o equipo de desarrollo no posee capital económico o logístico para mantener un servidor de manera local. Es decir, los sistemas severless representan una gran alternativa al modelo tradicional si se trata de un equipo de desarrollo pequeño. Por ejemplo, un equipo en proceso de emprendimiento o una pymes sin capital para mantener todo el modelo clásico del servidor.
- Debido a que React Native ofrece la posibilidad de desarrollar aplicaciones de múltiple plataforma con interfaces graficas iguales usando el mismo código, los costos de implementación son más bajos que tener dos equipos por separado, uno para Android y otro para IOS. Por ende, si una empresa cuenta con el presupuesto, puede manejar dos equipos, de lo contrario resulta más eficiente capacitar un equipo en React Native.
- Teniendo en cuenta que React Native es una tecnología muy moderna, todavía no cuenta con una comunidad lo suficientemente grande como para garantizar que todas las herramientas de programación estén implementadas y tengan un equipo formal que las mantenga. Debido a lo anterior, al momento de escoger React Native como entorno de desarrollo, es muy importante identificar todas las herramientas necesarias para implementar la aplicación; si todas las herramientas tienen un buen equipo de soporte en la web, React Native es el camino más apropiado para su implementación.
- Una vivienda del futuro debe tener la capacidad no solo de controlarse y monitorearse sino que también debe considerar su impacto ambiental, es decir, las casas del futuro deben transmitir la importancia de adquirir costumbres más amigables con el medio ambiente.

## **9.2. Trabajos futuros.**

Los servicios orientados a “IoT” se encuentran en crecimiento y cada vez más industrias se suman a la automatización de procesos basados en servicios orientados a internet. Por ende, la demanda de sistemas y modelos de comunicación escalables y seguros se verá incrementada de manera sostenible, y modelos como el implementado en este documento jugarán un papel importante en la transformación de estas industrias.

### **9.2.1. Pruebas con más dispositivos.**

Debido a los límites de este proyecto de grado, y las pruebas realizadas para comprobar su funcionamiento; el trabajo futuro más relevante para extender el alcance de este proyecto es conectar más dispositivos al sistema. El anterior proceso involucraría utilizar los dos clientes ya implementados (el aplicativo móvil y el de escritorio) para crear una red de usuarios que pueda usar todas las capacidades del sistema. Como aclaración, este proceso involucraría gastos asociados a la cantidad de almacenamiento utilizada y las cuotas de flujos de datos mínimas permitidas para que el sistema se mantenga gratis en el proveedor de servicios de nube utilizado.

### **9.2.2. Integración de sensores y actuadores al sistema.**

Teniendo en cuenta los objetivos del proyecto y como están encaminados a la competencia “Solar Decathlon Latinoamerica”, un trabajo futuro es integrar los sensores y actuadores que le permitan al sistema aprovechar todas sus características en la vivienda del Solar Decathlon; es decir, implementar desarrollos de hardware que realicen la medición real de las variables importantes para la competencia como: consumo eléctrico, consumo de agua, temperatura y humedad y velocidad del viento.

### **9.2.3. Interacción con la interfaz de voz de Google usando “Dialog flow”.**

Finalmente, para complementar el componente de inteligencia de la vivienda, se podría integrar una interfaz por voz que cumpla con las siguientes características: que sea fácil de usar para el usuario, que sea familiar a su contexto, que le permita al usuario hacer cambios en los actuadores del sistema, que le permita conocer el estado actual de los sensores y que los comandos de voz se puedan utilizar de manera natural.

## Referencias

- [1] S. Overflow. (2017, dec) Developer survey results 2017. [Online]. Available: <https://insights.stackoverflow.com/survey/2017>
- [2] F. Richter. (2019, jul) Amazon leads the race to the cloud. [Online]. Available: <https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/>
- [3] K. W. Kurose, J. F., & Ross, "REDES DE COMPUTADORAS Un enfoque descendente," *PEARSON Educación*, vol. 5, p. 793, 2010. [Online]. Available: <http://dialnet.unirioja.es/servlet/dcart?info=link&codigo=2741660&orden=170694>
- [4] A. Inc. (2019, aug)
- [5] Z. B. Babovic, J. Protic, and V. Milutinovic, "Web Performance Evaluation for Internet of Things Applications," *IEEE Access*, vol. 4, pp. 6974–6992, 2016.
- [6] K. Bhatia, "Nate Taggart on Serverless," *IEEE Software*, vol. 35, no. 4, pp. 101–104, 2018.
- [7] M. Amelung, K. Krieger, and D. Rösner, "E-assessment as a service," *IEEE Transactions on Learning Technologies*, vol. 4, no. 2, pp. 162–174, 2011.
- [8] J. E. Giral Sala, R. Morales Caporal, E. Bonilla Huerta, J. J. Rodriguez Rivas, and J. D. J. Rangel Magdaleno, "A Smart Switch to Connect and Disconnect Electrical Devices at Home by Using Internet," *IEEE Latin America Transactions*, vol. 14, no. 4, pp. 1575–1581, 2016.
- [9] F. M. Douglas Wilson, Andrey Sidorov. (2019, aug)
- [10] D. W. Bartosz Skuza, Agnieszka Mroczkowska. (2019, sep) Flutter vs react native – what to choose in 2019. [Online]. Available: <https://www.thedroidsonroids.com/blog/flutter-vs-react-native-what-to-choose-in-2019>
- [11] R. Shete and S. Agrawal, "IoT Based Urban Climate Monitoring using Raspberry Pi," *International Conference on Communication and Signal Processing, April 6-8, 2016, India IoT*, pp. 2008–2012, 2016.
- [12] A. Howedi and A. Jwaid, "Design and implementation prototype of a smart house system at low cost and multi-functional," *FTC 2016 - Proceedings of Future Technologies Conference*, no. December, pp. 876–884, 2017.

- [13] A. Imteaj, T. Rahman, M. K. Hossain, M. S. Alam, and S. A. Rahat, "An IoT based Fire Alarming and Authentication System for Workhouse using Raspberry Pi 3," *ECCE 2017 - International Conference on Electrical, Computer and Communication Engineering*, no. 0, pp. 899–904, 2017.
- [14] J. Cabrera, M. Mena, A. Parra, and E. Pinos, "Intelligent assistant to control home power network," *2016 IEEE International Autumn Meeting on Power, Electronics and Computing, ROPEC 2016*, no. Ropec, 2017.
- [15] B. Y. B. Chacos, "Raspberry Pi 3 : The cures its biggest headaches."
- [16] N. Hossain, M. T. Kabir, T. R. Rahman, M. S. Hossen, and F. Salauddin, "A real-time surveillance mini-rover based on OpenCV-Python-JAVA using Raspberry Pi 2," *Proceedings - 5th IEEE International Conference on Control System, Computing and Engineering, ICCSCE 2015*, no. November, pp. 476–481, 2016.
- [17] J. C. Shovic, *Raspberry Pi IoT Projects Prototyping Experiments for Makers Raspberry Pi IoT Projects Prototyping Experiments for Makers Raspberry Pi IoT Projects: Prototyping Experiments for Makers*, 2016. [Online]. Available: <https://link.springer.com.zorac.aub.aau.dk/content/pdf/10.1007%2F978-1-4842-1377-3.pdf>
- [18] B. Nakhuva and T. Champaneria, "Study of Various Internet of Things Platforms," *International Journal of Computer Science & Engineering Survey*, vol. 6, no. 6, pp. 61–74, 2016.
- [19] C. Herrera, J. Simon, C. E. Rodriguez, A. F. Jaramillo, A. Bernal, S. Ospina, and M. Luna, "Solar Decathlon Latin America and caribbean," Tech. Rep. 1967, 2015.