



**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
14 de Agosto de 2019



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

JUAN DAVID RAMÍREZ VILLEGAS
CODIGO: 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

14 de Agosto de 2019

Índice

Resumen	6
1 Marco Teórico	8
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	9
1.7 Mqtt	10
1.8 Http	10
1.9 Serverless	11
1.10 Backend	11
1.11 Frontend	11
1.12 Dispositivo	12
1.13 Teoría de colores	12
2 Introducción	14
2.1 Contexto introductorio	14
2.2 Lineamientos de desarrollo	14
2.3 Diseño Generalizado	16
3 Api de animación	19
3.1 Investigación y conceptualización	19
3.2 Api de java	20
3.3 Api de Python y Qt	24
4 House Manager	27
4.1 Especificaciones técnicas	27
4.2 Requerimientos Funcionales	28
4.3 Funciones adicionales	30
4.4 Algoritmos Utilizados	32
4.4.1 Scheduler Handler	33
4.4.2 Remote Mode Handler	33
4.4.3 Mail Notification Handler	33
4.4.4 Data Report Handler	33
4.4.5 Indicator Handler	33
5 Serverless backend	34
5.1 Especificaciones técnicas	35
5.2 Base de datos relacional y no relacional	35

5.3	Relación de requerimientos funcionales	39
5.4	Funciones de nube	40
5.4.1	Get_historical_data:	40
5.4.2	Send_command:	43
5.4.3	IOT_gateway_control:	46
5.4.4	IOT_gateway_measure:	47
6	User app	49
6.1	Especificaciones técnicas	49
6.2	Requerimientos Funcionales	52
6.3	Funciones adicionales	55
6.4	Pruebas de concepto	57
7	Conclusiones y Trabajos futuros	59
7.1	Conclusiones	59
7.2	Trabajos futuros.	60
7.2.1	Pruebas con más dispositivos.	60
7.2.2	Integración de sensores y actuadores al sistema.	60
7.2.3	Interacción con la interfaz de voz de Google usando “Dialog flow”.	60
	Referencias	61

Índice de figuras

Figura 1	Diagrama generalizado de las estructuras de software. Fuente: propia	18
Figura 2	Conceptualización inicial del api en sitio. Fuente: propia	20
Figura 3	Estadísticas de los lenguajes de programación con más demanda. Fuente: [1]	21
Figura 4	El diseño de estados del botón interactivo. Fuente: propia . . .	22
Figura 5	El diseño de los estados del botón interactivo desplegable. Fuente: propia	22
Figura 6	Movimiento según la ubicación de la función horizontal. Fuente: propia	22
Figura 7	Primer Aproximación del diseño de la interfaz en sitio. Fuente: propia	23
Figura 8	Ajuste de diseño de la interfaz en sitio. Fuente: propia	23
Figura 9	Versión final del concepto de diseño. Fuente: propia	24
Figura 10	Botón de tamaño flexible genérico de la api de qt. Fuente: propia	25
Figura 11	Boton de tamaño fijo usando imagenes y slider. Fuente: propia	26
Figura 12	Visualización de la escalabilidad del sistema con una app no desarrollada por completo. Fuente: propia	28
Figura 13	Escena del home del House Manager con barras de progreso para la indicación dinámica. Fuente: propia	29
Figura 14	Escena de control del House Managern con sus calendarios. Fuente: propia	30
Figura 15	Escena de medición del House Manager, con su respectivo panel para guardar los datos. Fuente: propia	31
Figura 16	Escena de configuración del dispositivo House Manager. Fuente: propia	31
Figura 17	Escena para el inicio de sesión del House Manager. Fuente: propia	32
Figura 18	Diseño de las tablas de la base de datos relacional MySql. Fuente: propia	36
Figura 19	Grupo de tablas: users data. Fuente: propia	37
Figura 20	Grupo de tablas: device info	38
Figura 21	Grupo de tablas: device data. Fuente: propia	39
Figura 22	Algoritmo de la función: Get historical data. Fuente: propia . .	42
Figura 23	Algoritmo de la función: Send command. Fuente: propia	45
Figura 24	Algoritmo de la función: Iot gateway control. Fuente: propia .	47
Figura 25	Algoritmo de la función: Iot gateway measure. Fuente: propia .	48
Figura 26	El menú desplegable con las aplicaciones actualmente desplegadas. Fuente: propia	51
Figura 27	Escena para la medición en tiempo real de las variables. Fuente: propia	53

Figura 28	Escena de configuraciones, actualmente no disponible. Fuente: propia	54
Figura 29	Escena para la visualización del consumo histórico. Fuente: propia	54
Figura 30	Escena para el control de los circuitos de la vivienda. Fuente: propia	56
Figura 31	Escena de bienvenida 1. Fuente: propia	56
Figura 32	Escena de inicio de sesión. Fuente: propia	57
Figura 33	Dispositivo Inelca utilizado para la prueba de concepto. Fuente: propia	58
Figura 34	Prueba de concepto usando todos los sistemas. Fuente: propia .	58

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 una arquitectura software, 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 principalmente de manejar los sensores y actuadores de manera local, en caso de pérdida de conexión, y de manera remota en presencia de una red wifi. 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. Pero por el alcance del proyecto no se realizaron pruebas asociadas a esta característica.

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 software architecture 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 responsible, in a general manner, for monitoring and controlling remotely all devices connected to the system. The desktop application was mainly responsible for handling the sensors and actuators locally, in the case of a lost connection, and remotely in the presence of a Wi-Fi network. Finally, the backend service was responsible for communicating both frontend products and storing user information.

Taking into account the above, the architecture was designed with the possibility of supporting up to 4000, including computers, cell phones and any device capable of executing routines in Python or Nodejs. But due to the scope of the project, no tests associated with this characteristic were performed.

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 en 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.

La arquitectura de software que se implementó en este proyecto 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 web puede ser una aplicación de escritorio. 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

Un módulo hardware y software con amplia capacidad de almacenamiento y procesamiento. 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 con especificaciones de hardware fijas como: RAM, CPU, memoria interna, etc, enfocado únicamente en ofrecer los servicios anteriormente mencionados.

1.5. Tcp y Tls

Tcp (Transmission 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”[2]. 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 (JavaScript object notation) para crear un token que sirvan para enviar datos entre aplica-

ciones 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 [3]

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 [4]. 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 concentre en desarrollar aplicaciones y administrar el estado de las aplicaciones” [5].

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 [6]. 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 necesarios 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 de la arquitectura 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 retro alimenta 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

No es para nadie una sorpresa que durante este siglo, las nuevas generaciones que nacen rodeadas de la tecnología y 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 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 [7], por otra parte, tenemos beneficios de monitorización y la capacidad de añadir herramientas que permitan la integración con redes eléctricas inteligentes. 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.

 Universidad del Valle	Universidad del Valle –Implementación de un gestionado de recursos para el hogar con inteligencia ambiental. —	Rev: 003
Título: ESPECIFICACIÓN DE REQUERIMIENTOS FUNCIONALES	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.	E
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.

2.3. Diseño Generalizado

El sistema se diseñó con 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 desarrollaron aplicaciones de frontend capaces de integrarse con 2 tipos de protocolos de comunicación: Mqtt y Https; además, para acceder a los servicios de la nube se implementó el servicio de backend en la plataforma “Google Cloud Plataform” puesto que permite 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.

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 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 serverless le permitieron a la aplicación ser diseñada para permanecer funcional sin hacer ningún cambio inclusive con 4000 usuarios (esta última apreciación es teórica puesto que no se hicieron pruebas a esa escala).

Para comprender mejor el producto, se puede observar en la figura 1 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).

El producto del backend se implementó en el servicio de Google Cloud Services y utilizó los siguientes elementos para su funcionamiento: Un “Storage version 3 system” 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 http 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 Mqtt de los dispositivos del sistema.

La aplicación móvil desarrollada está 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 desarrolló usando Python y se diseñó para manejar un flujo de datos más grandes y la posibilidad de funcionar desconectado de la red. La aplicación le permitió 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.

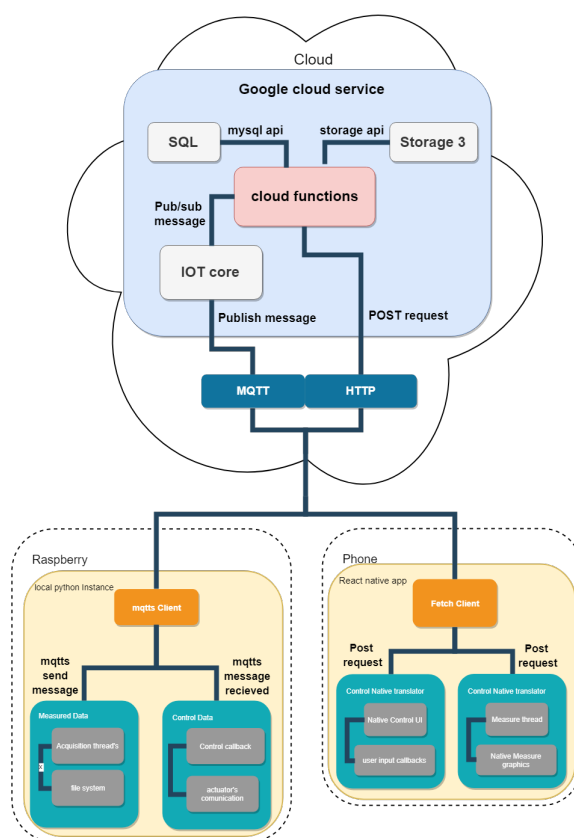


Figura 1: Diagrama generalizado de las estructuras de software. Fuente: propia

3. Api de animación

3.1. 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, Processing, 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 2). 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.

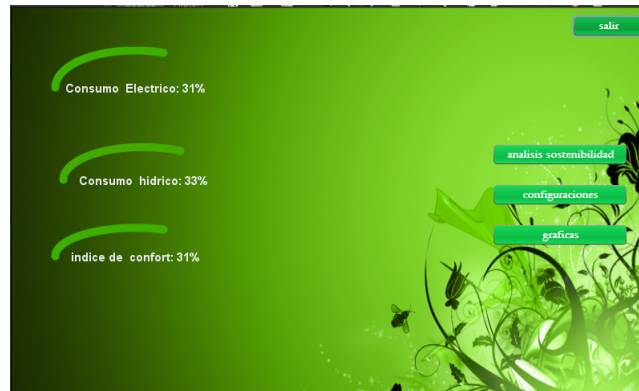


Figura 2: Conceptualización inicial del api en sitio. Fuente: propia

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.

En adición, 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 3. Pero esta característica se debe principalmente a que el sistema bancario tradicional esta construido sobre servicios de backend utilizando Java.

3.2. 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 4. Este objeto funciona

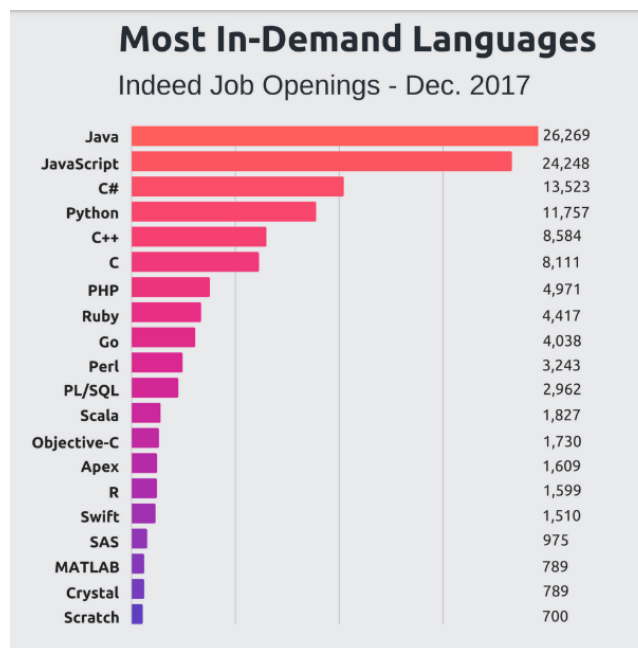


Figura 3: Estadísticas de los lenguajes de programación con más demanda. Fuente: [1]

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 grafico 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. Un implementación utilizada en la aplicación se puede observar en la figura 5. 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 6. La función permite modificar la ubicación actual del objeto moviéndolo de manera linear con una rata de refresco dada (actualización visual) y una velocidad constante, desde una coordenada X o Y dependiendo de en cual eje coordenado se desee 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 7, debido a la recursiva implementación grafica se obtuvo un programa que sobrecargaba un sistema de cómputo completo (portátil Dell con procesador i5 segunda

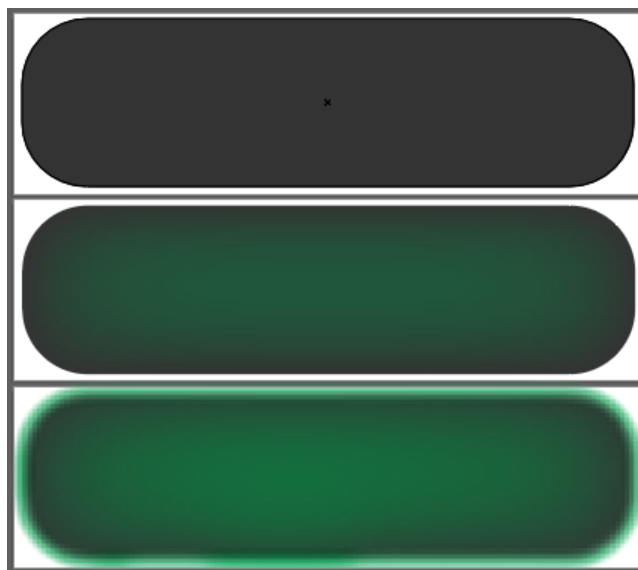


Figura 4: El diseño de estados del botón interactivo. Fuente: propia

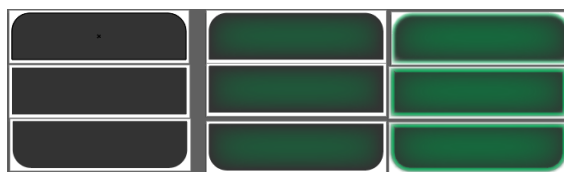


Figura 5: El diseño de los estados del botón interactivo desplegable. Fuente: propia



Figura 6: 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).

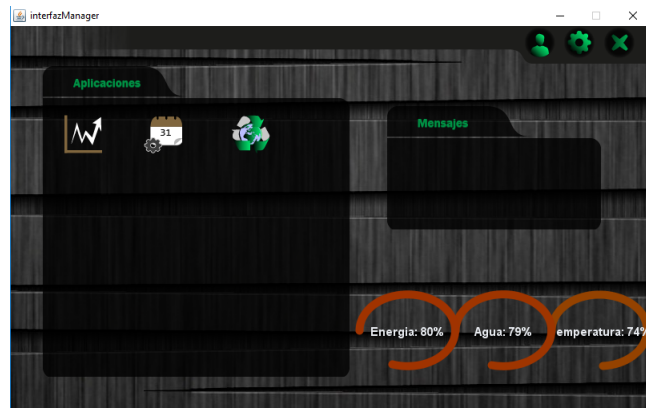


Figura 7: Primer Aproximación del diseño de la interfaz en sitio. Fuente: propia

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 8 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 8: 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 9. 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).

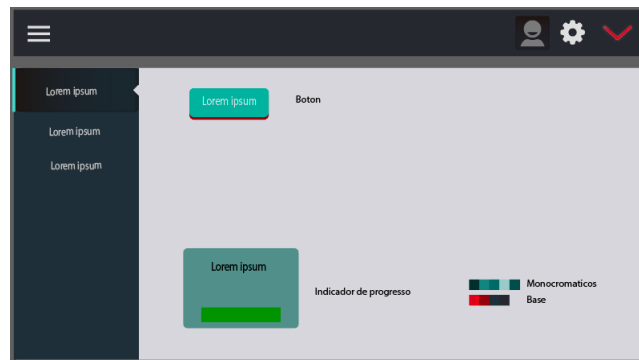


Figura 9: Versión final del concepto de diseño. Fuente: propia

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 de la api 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 se trataba de un módulo completamente escrito en C le permitía a la aplicación tener un mejor rendimiento gráfico, con la desventaja de dificultar el efecto multiplataforma de Python, para mantener el efecto, el framework debería soportar por completo la plataforma sobre la cual se estuviera corriendo (Windows, macos, Linux, Ubuntu, etc).

3.3. 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 estatico del aplicativo se realiza a partir de un “lenguaje de marcado” (markup language) autogenerado llamado Qml que define las propiedades graficas 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, facilito la implmentacion de la api puesto que se relacionaba directamente con la forma como se desarrolla frntend 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 logro 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 10.



Figura 10: 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 11.

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 11.

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.

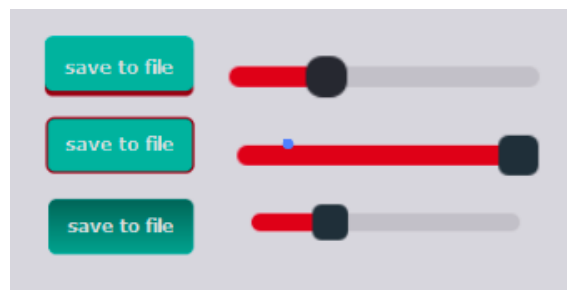


Figura 11: Boton de tamaño fijo usando imagenes y slider. Fuente: propia

4. 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” del formato RUP 1. Para el desarrollo de estos componentes fueron necesarias diferentes condiciones técnicas que serán descritas en el siguiente apartado.

4.1. 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 librerías 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 gráfico 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*.

Dentro de la carpeta del proyecto 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.

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 12. 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 próximos apartados.

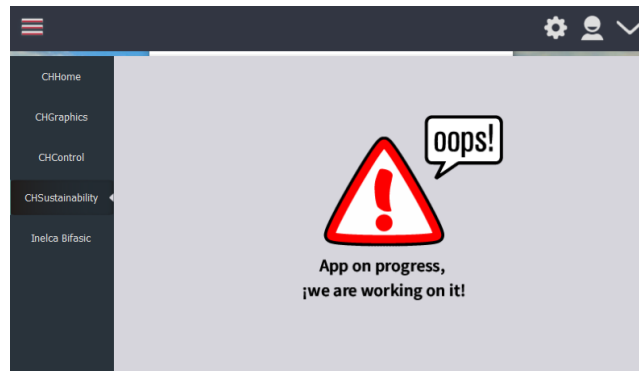


Figura 12: Visualización de la escalabilidad del sistema con una app no desarrollada por completo. Fuente: propia

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

4.2. Requerimientos Funcionales

Durante la etapa de conceptualización del proyecto de ingeniería que este documento describe se utilizó una herramienta de planeación llamada RUP; con un documento llamado “Requerimientos funcionales”, dentro de los requerimientos implementados descritos en ese documento están:

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 vivienda¹³. 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

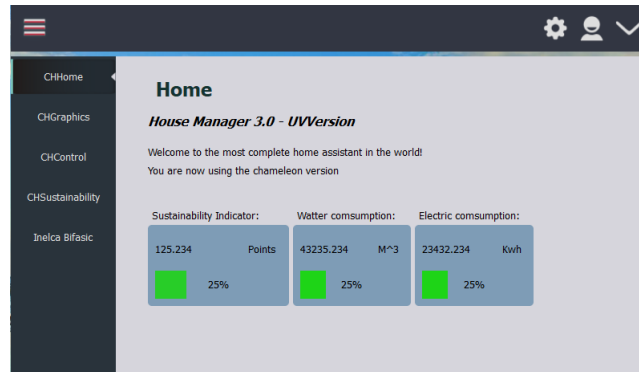


Figura 13: Escena del home del House Manager con barras de progreso para la indicación dinámica. Fuente: propia

funcionalidad se utilizó un elemento gráfico tipo calendario para configurar de manera individual las ventanas de activación de las cargas eléctricas en la vivienda¹⁴.

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.

En adición, 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 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 encriptacion del broker de Mqtt (para más información de los endpoints y la rutina de encriptacion vease el capítulo *backend*).

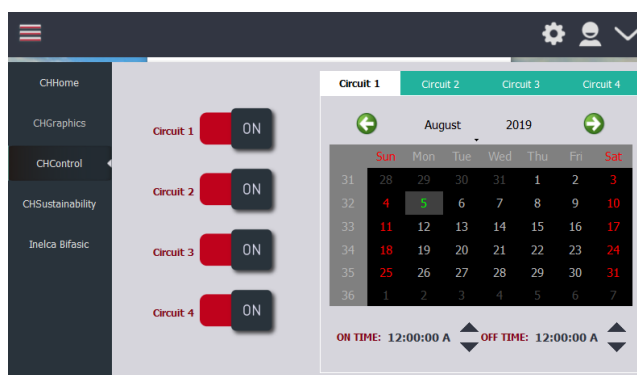


Figura 14: Escena de control del House Managern con sus calendarios. Fuente: propia

4.3. 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 de la adquisición. En caso de la caída repentina de la red es capaz de cambiar al modo local sin la pérdida de ningún dato. También, es capaz de entender cuando el usuario desea que el dispositivo funcione permanentemente de manera local.
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. 15.
3. **Ajustes de configuración** Esta es una característica que incluye una ventana adicional donde se configuran los aspectos concernientes al sistema en general¹⁶. 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 cubico 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).

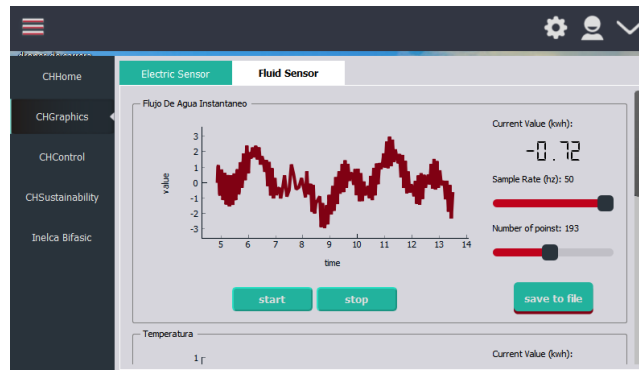


Figura 15: Escena de medición del House Manager, con su respectivo panel para guardar los datos. Fuente: propia

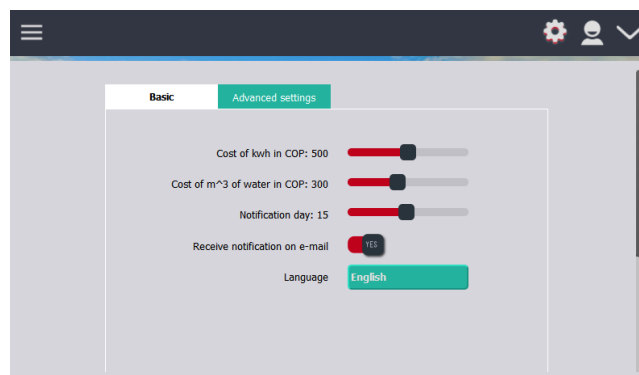


Figura 16: Escena de configuración del dispositivo House Manager. Fuente: propia

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 17.

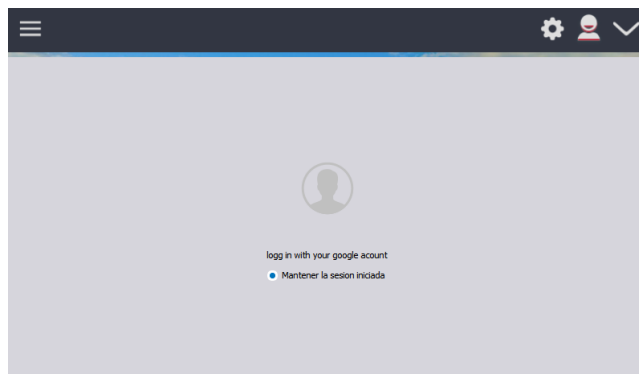


Figura 17: Escena para el inicio de sesión del House Manager. Fuente: propia

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.

4.4. Algoritmos Utilizados

Los algoritmos utilizados en esta aplicación representan un hilo de programa en Python y 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.

4.4.1. Scheduler Handler

Este Hilo es el encargado de verificar y mantener el estado de los circuitos en el establecido en el horario de configuración. Este hilo de programa es capaz de discernir los circuitos que se han activado de manera manual desde la aplicación o desde la interfaz, con el objetivo de desactivar por 24 horas la influencia del calendario configurado. .

4.4.2. 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 pérdida de información.

4.4.3. 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.

4.4.4. 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 pérdida para variables de medición con una granularidad mayor a 1 segundo.

4.4.5. 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.

5. 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 superficialmente las siguientes opciones: Amazon Web Services, Google Cloud y Microsoft Azure. Con una primera mirada se pudo observar que los factores diferenciadores de cada proveedor dependían estrictamente de los productos adicionales que estos podían ofrecer.

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 línea y sus servicios de storage webs. En resumen, su especialidad es el servicio de almacenamiento. Teniendo esto en mente, resultó una tentativa debido a la necesidad de nuestro proyecto de almacenar información de muchos dispositivos.

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 acceder a servicios clave para proyectos como este como el bróker serverless de IOT.

Finalmente, Microsoft Azure es el servicio de nube con la posibilidad de integrarse de la mejor manera con desarrollos .Net, lo anterior debido a que es la tecnología de desarrollo creada y administrada por Microsoft.

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, principalmente porque poseía la implementación del bróker de Mqtt más apropiada para el desarrollo de un proyecto de estas características. Además, el servicio de Google permitiría utilizar herramientas propias como su interfaz de procesamiento de voz natural, en el momento que se extienda el alcance del proyecto.

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”.

5.1. Especificaciones técnicas

Para el montaje de la arquitectura 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 para como bróker de internet de las cosas y las “Cloud Functions” de su arquitectura de nube. De los elementos anteriormente mencionados se puede entender el sistema 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. Un diagrama generalizado se puede ver en el capítulo de introducción en la figura 1.

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 esta arquitectura se decidió utilizar el entorno de desarrollo de Node.js, Por ende para poder hacer uso de los componentes necesarios para implementar la arquitectura planeada 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.

5.2. Base de datos relacional y no relacional

Una parte muy importante de la arquitectura 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 es la base de datos relacional MySQL.

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 18, donde cada cuadro representa un grupo de tablas bajo el paradigma SQL.

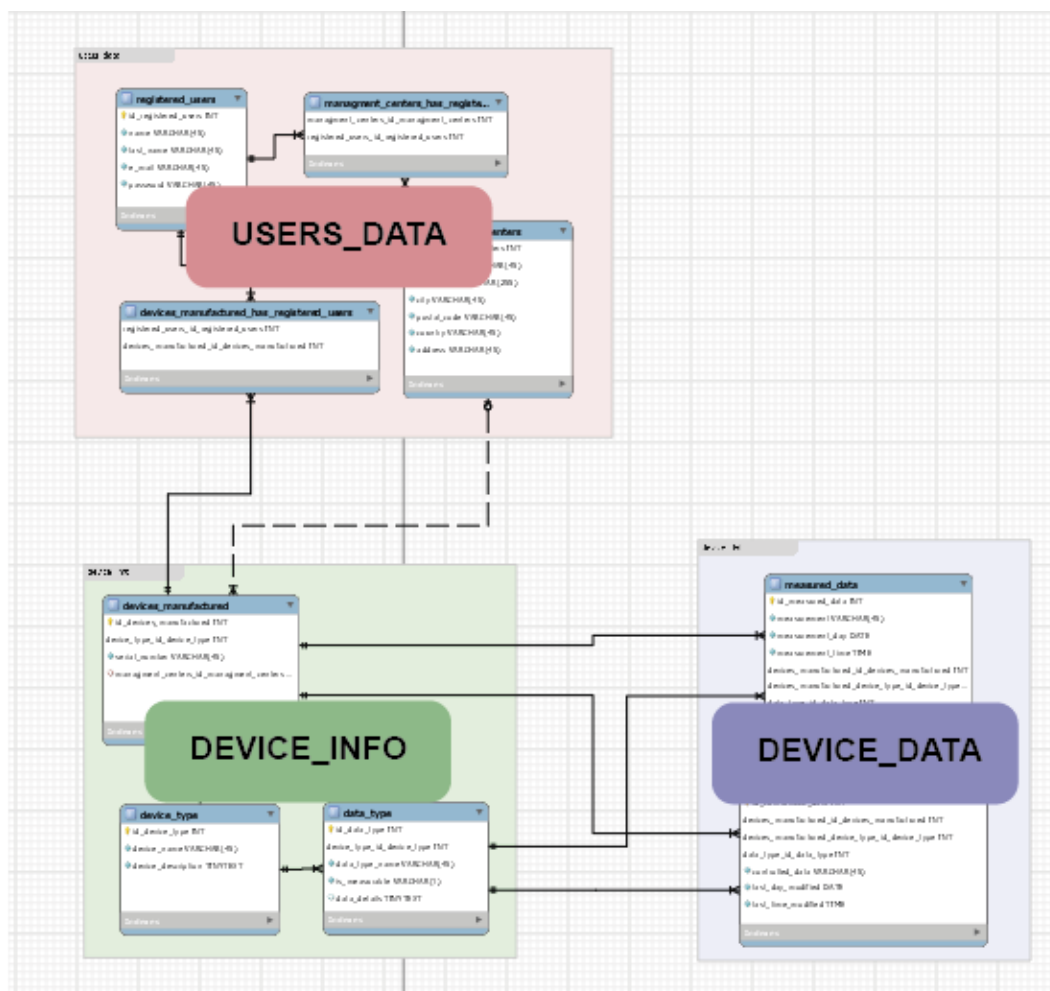


Figura 18: Diseño de las tablas de la base de datos relacional MySQL. Fuente: propia

Teniendo en cuenta lo anterior, se explicará por separado cada grupo de tablas

con el objetivo de mostrar la escalabilidad del sistema y como fue implementada la base de datos pensando en los requerimientos funcionales de los productos House Manger y User App. cabe aclarar que para representar los tipos de columnas 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 las columnas sin símbolo son llaves secundarias de relación obligatorias.

- **Users data:** 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 `managment_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. En el caso de realiza envíos o referenciar espacios geográficos esta tabla presenta una utilidad más representativa.

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

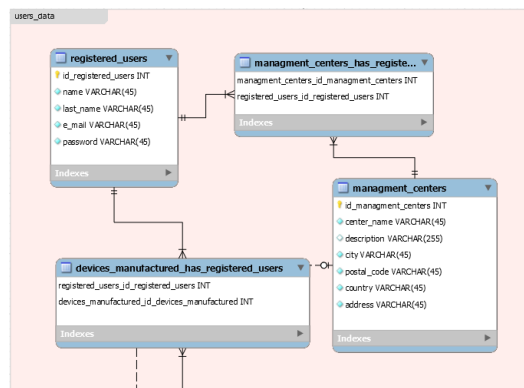


Figura 19: Grupo de tablas: users data. Fuente: propia

- **Device info:** En este grupo existen 3 tablas y todas contienen información relevante: La tabla `devices_manufactured` contiene principalmente el número serial. 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. 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.

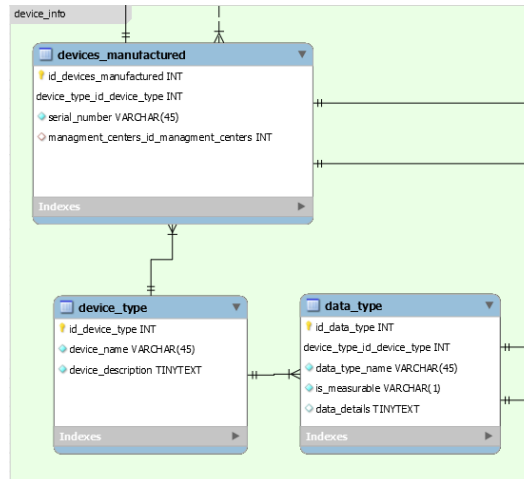


Figura 20: Grupo de tablas: device info

- **Device_data:** El grupo de tablas de información se compone únicamente de 2; la tabla `measured_data` contiene únicamente 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. Para la versión de la base de datos que se está utilizando se pueden permitir 4000

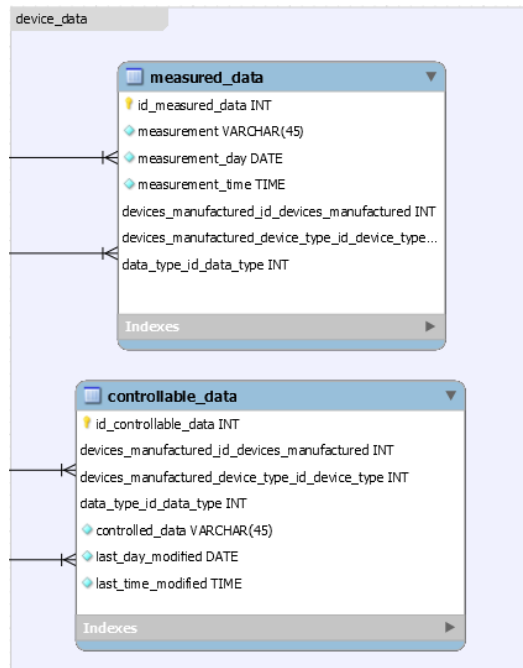


Figura 21: Grupo de tablas: device data. Fuente: propia

conexiones simultaneas, por ende, se dice que el sistema está diseñado para 4000 usuarios.

5.3. Relación de requerimientos funcionales

Debido a que este producto no se contempló en la etapa 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 gastos 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.

5.4. Funciones de nube

Las funciones de nube implementadas 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.

Teniendo en cuenta lo anterior y los requerimientos funcionales se implementaron 4 funciones de nube que serán explicadas en los siguientes apartados.

5.4.1. 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 Sting encriptado conteniendo el objeto de autenticación. El Sting 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 22, 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 unicamente 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 obtener la llave “`verificadora`” del usuario en cuestión.
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.

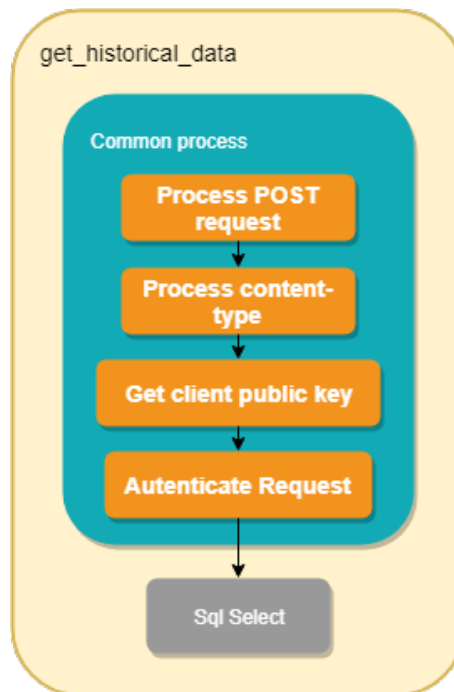


Figura 22: Algoritmo de la función: Get historical data. Fuente: propia

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 Sting. 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).

5.4.2. 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 enviara un mensaje al dispositivo, si es posible, y finalizara 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", "date" y "time" contienen 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 columnas 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 [8].

Adicionalmente, un diagrama generalizado del algoritmo se puede ver en la figura 23. 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. **Adquisición de la llave publica:** También, hacemos la consulta al servicio de almacenamiento para adquirir la llave pública que será usada en el proceso de 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")
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.

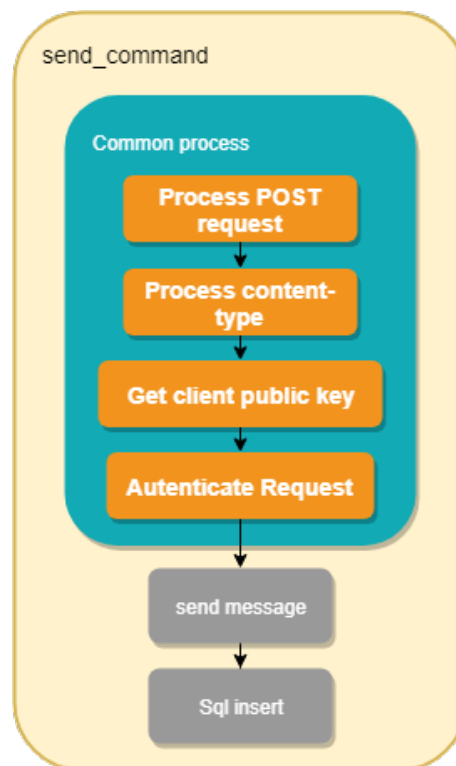


Figura 23: Algoritmo de la función: Send command. Fuente: propia

5.4.3. 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 “encrypted.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 24. 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.
2. **Discriminación de operacion:** Asi como se discriminó en la funcion “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

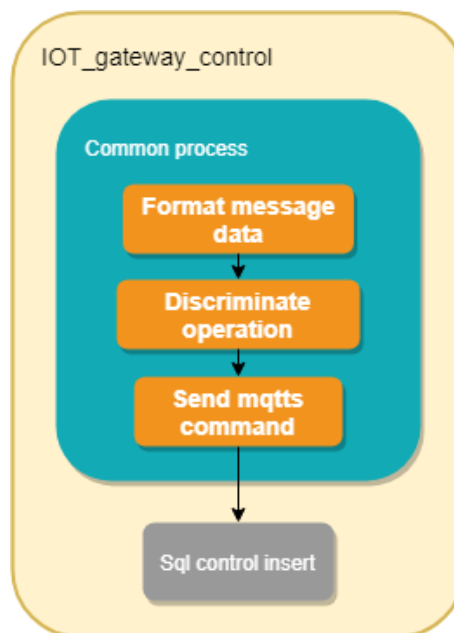


Figura 24: Algoritmo de la función: Iot gateway control. Fuente: propia

3. **Envío del mensaje de control Mqtt:** Se envía exactamente el contenido del campo mensaje al dispositivo de interés a través 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.

5.4.4. 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 25. 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.

En adición, 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 Sting que identifique el tópico Mqtt que el dispositivo destino pueda reconocer como un mensaje de control.

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.

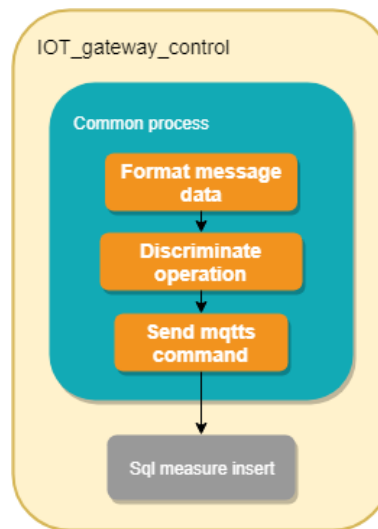


Figura 25: Algoritmo de la función: Iot gateway measure. Fuente: propia

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*.

6. User app

Teniendo en miras de la escalabilidad deseada para este sistema se escogió el framework de desarrollo de aplicaciones móviles llamado React Native. Este es un framework de JavaScript desarrollado y mantenido por Facebook Inc con ventajas significativas desde la perspectiva de diseño: 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.

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

6.1. 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.
- **Jrsrsign:** 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 26.
- **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.

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 “snippet” (Archivo o script que cumple una función en específico) era capaz de funcionar en el entorno de programación de React-Native. Este pequeño “snippet” 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.

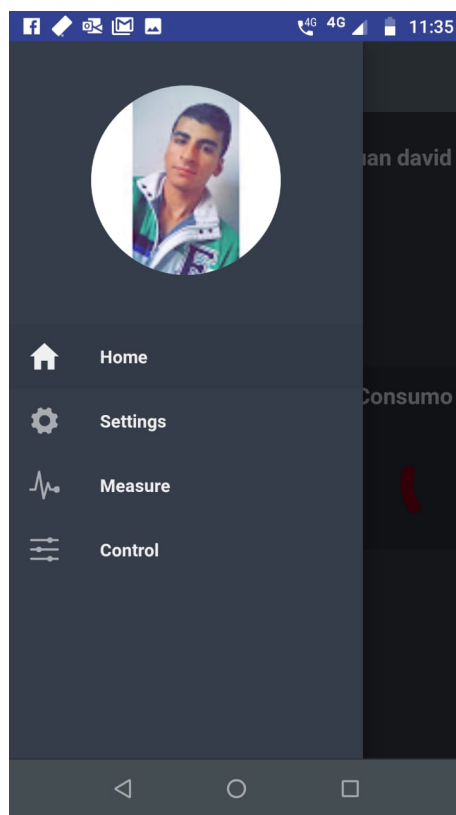


Figura 26: El menú desplegable con las aplicaciones actualmente desplegadas. Fuente: propia

6.2. Requerimientos Funcionales

Debido a que durante el proceso de planeación del proyecto también se utilizó la herramienta de RUP 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 funcionaria 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 27, donde deslizándose hacia abajo se puede tener acceso a las gráficas con contenido de otros sensores.

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 28.

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 32 donde el usuario puede panear horizontalmente el centro de la pantalla para revelar los otros indicadores.



Figura 27: Escena para la medición en tiempo real de las variables. Fuente: propia

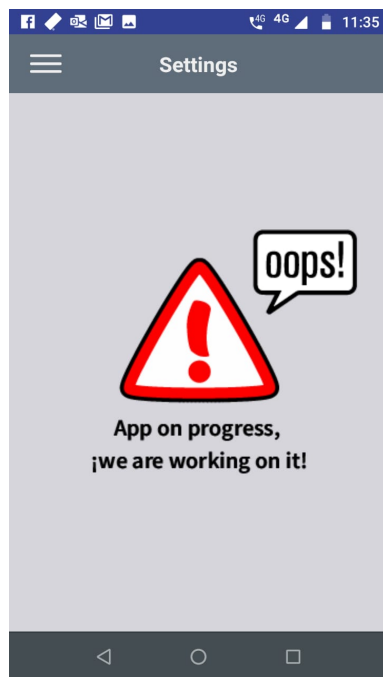


Figura 28: Escena de configuraciones, actualmente no disponible. Fuente: propia

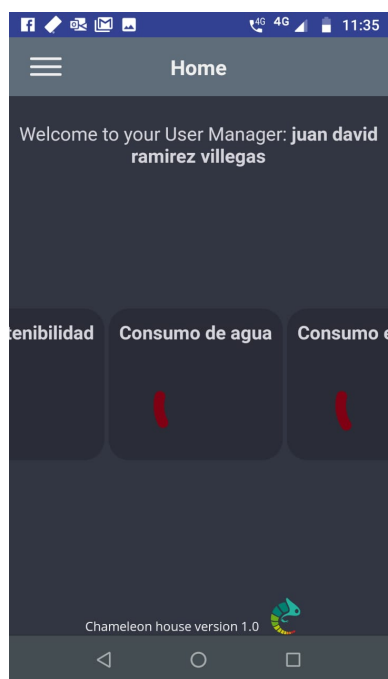


Figura 29: Escena para la visualización del consumo histórico. Fuente: propia

4. *“La aplicación notificara al usuario cuando la factura este vencida:”* para este requerimiento se implementó una función adicional en el aplicativo en sitio, por ende, para esta versión de la aplicación no se desarrolló ningún componente relacionado con esta necesidad.
5. *“El usuario debe ser capaz de poder ver su impacto ambiental basado en datos cualitativos y cuantitativos:”* Teniendo en cuenta la figura 29 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 no fue implementado para esta versión de la aplicación.
6. *“El sistema debe ser capaz de notificar las pérdidas 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.

6.3. 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 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 30, 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 31.
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 32.

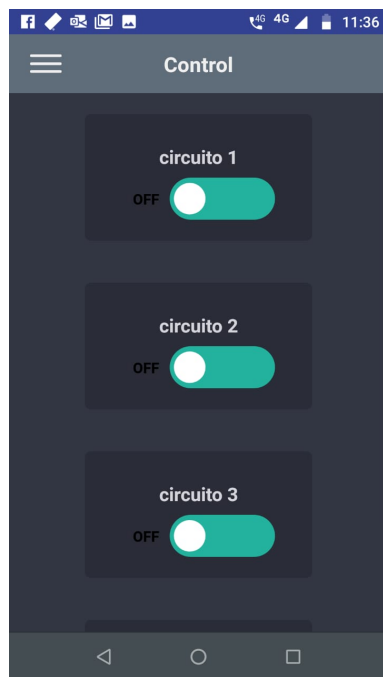


Figura 30: Escena para el control de los circuitos de la vivienda. Fuente: propia

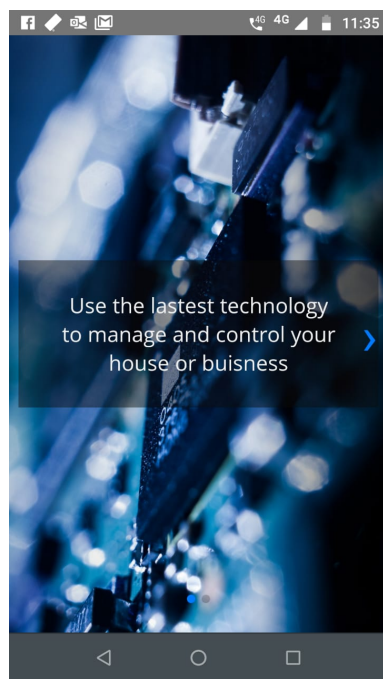


Figura 31: Escena de bienvenida 1. Fuente: propia

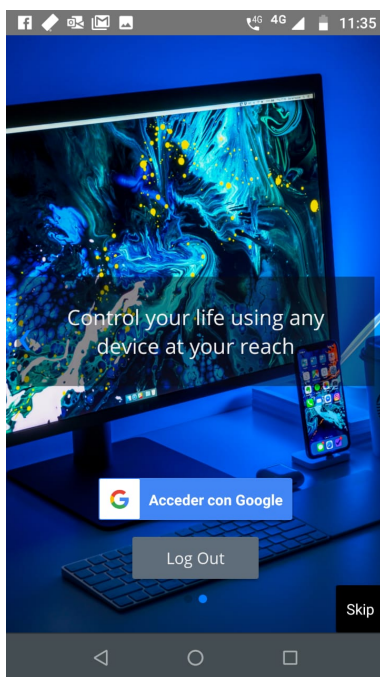


Figura 32: Escena de inicio de sesión. Fuente: propia

6.4. Pruebas de concepto

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.

Para finalizar, se puso a prueba el sistema añadiendo la función de adquisición necesaria para obtener el valor del voltaje Rms del medidor a partir de una trama serial tipo modbus, tal como se observa en la figura 34.



Figura 33: Dispositivo Inelca utilizado para la prueba de concepto. Fuente: propia



Figura 34: Prueba de concepto usando todos los sistemas. Fuente: propia

7. Conclusiones y Trabajos futuros

7.1. 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 entorno a mejorar el sistema de interfaces es 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 framework 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.

7.2. 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.

7.2.1. 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.

7.2.2. 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.

7.2.3. 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, dec 2017.
- [2] 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>
- [3] A. Inc., aug 2019.
- [4] Z. B. Babovic, J. Protic, and V. Milutinovic, “Web Performance Evaluation for Internet of Things Applications,” *IEEE Access*, vol. 4, pp. 6974–6992, 2016.
- [5] K. Bhatia, “Nate Taggart on Serverless,” *IEEE Software*, vol. 35, no. 4, pp. 101–104, 2018.
- [6] 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.
- [7] 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.
- [8] F. M. Douglas Wilson, Andrey Sidorov, aug 2019.
- [9] 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.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] B. Y. B. Chacos, “Raspberry Pi 3 : The cures its biggest headaches.”

- [14] 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.
- [15] 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%7B%7D2F978-1-4842-1377-3.pdf>
- [16] 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.
- [17] 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.